UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale in Progettazione e Sviluppo Sistemi Software

Strumenti e tecniche di testing automation per applicazioni di realtà aumentata

Anno Accademico 2019/2020

Relatore
Ch.mo Prof.ssa Anna Rita Fasolino

Correlatore
Ch.mo Prof. Porfirio Tramontana

Candidato
Sabato Danilo Bevilacqua
matr. M63000836

Per aspera sic itur ad astra

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione alla Realta' Aumentata

In questo capitolo chiariremo il significato di realtà aumentata (AR, dall'inglese Augmented Reality), le sue origini e la sua evoluzione attraverso i decenni. Durante questo percorso vedremo i devices che ne hanno permesso l'utilizzo. Sarà inoltre fatta una differenziazione tra AR, realtà mista (MR, dall'inglese Mixed Reality) e realtà virtuale (VR, dall'inglese Virtual Reality). Infine saranno presentati alcuni esempi di applicazioni AR.

1.1 Definizione

Secondo il dizionario di Oxford la realtà aumentata è definita: *"a technology that combines computer-generated images on a screen with the*

real object or scene that you are looking at" [3], quindi si può affermare che l'AR ha come obiettivo l'arricchimento del mondo reale, sovrapponendo ad esso un insieme di dati virtuali con cui è possibile interagire. Più precisamente, le caratteristiche principali di un sistema/applicazione AR sono: combinazione di oggetti virtuali nel mondo reale, interazione real-time e gestione di oggetti 3D [28].

1.2 Cenni storici

A Ivan Sutherland si deve la creazione del primo sistema di realtà aumentata, avvenuta nel 1968, che prevedeva l'utilizzo di un Head-Mounted Dispaly (HMD) cablato [48]. Per HMD s"intende una sorta di casco che l'utente indossa al fine di potersi calare nel mondo della realtà aumentata, grazie al collegamento cablato a una serie di computers. Un ulteriore passo avanti fu compiuto nel 1975, quando Myron Krueger creò "Videoplace", ossia una stanza che permetteva agli utenti di interagire con oggetti virtuali. Bisogna attendere il 1990 per la coniazione del termine "Realtà Aumentata" da parte di Tom Caudell [45]. Caudell coniò il termine dopo aver realizzato un sistema, deployato su HMD, per aiutare i lavoratori della Boeing a cablare gli aerei, visualizzando i loro schemi di cablaggio sul pavimento della fabbrica [56]. I successivi dieci anni videro un grande sviluppo dell'AR, tanto che nel 2000 Bruce Thomas creò il primo gioco AR per dispositivi mobili esterni, ARQuake [45]. Tuttavia, nonostante le applicazioni

AR fossero realizzate con successo, i costi della tecnologia su cui potevano essere deployate erano proibitivi, senza contare la conoscenza tecnica necessaria per utilizzarla. Tutto ciò aveva precluso al grande pubblico l'utilizzo di questa nuova e affascinante tecnologia. La vera e propria svolta, dal punto di vista hardware, si è avuta con l'avvento degli smartphone, che grazie a: fotocamera ad alta risoluzione, sensori, accelerometri e GPS hanno permesso al grande pubblico un modo facile, veloce ed economico per poter usufruire di applicazioni di realtà aumentata. Oltre agli smartphone sono utilizzate altre classi di dispositivi portatili come ad esempio:

- Tablet PC [52]: molto più potenti rispetto agli smartphone ma più costosi e ingombranti, tanto che l'utilizzo prolungato con una o due mani risulta stancante [45];

- Dispositivi indossabili: lenti a contatto, braccialetti wireless [53] ma anche gli stessi HMD. Questi ultimi hanno fatto passi da gigante, infatti possiedono una capacità di calcolo intrinseca tale da renderli privi di cablaggi ai computers e quindi del tutto wireless [35].

Ad oggi, come si è potuto constatare, i dispositivi su cui deployare applicazioni AR sono i più disparati. La scelta di un dispositivo piuttosto che un altro è dovuta alla stessa applicazione, in modo tale da fornire all'utente la migliore esperienza possibile.

1.3 MR e VR

Oltre all'AR, oggi sentiamo sempre più parlare di Realtà Mista (MR) e Realtà Virtuale (VR). Anch'esse sono tecnologie che integrano componenti virtuali e mondo reale, fornendo esperienze immersive. Cerchiamo di fare più chiarezza definendone i rispettivi significati.

1.3.1 Realtà Mista

Per MR s'intende una tecnologia che combina elementi digitali con quelli del mondo reale al fine di ottenere un mondo ibrido. Gli utilizzatori della MR possono interagire con i contenuti digitali, visualizzati nel mondo reale, attraverso: gesti, sguardi, riconoscimento vocale e via discorrendo. In questo modo l'utente della MR può tranquillamente interagire sia con l'ambiente reale che con quello digitale, poiché la differenza tra queste due tipologie di oggetti sarà quasi impercettibile. L'incapacità di distinzione tra oggetto reale e virtuale è dovuta alla visione a 360°, che fornirà le diverse angolazioni dello stesso quando ci si gira attorno, e al rispetto della prospettiva man mano che ci si avvicina o ci si allontana da esso. Per quanto riguarda i dispositivi, tra i più utilizzati troviamo HMD, smart-glass e proiettori. Questi ultimi sono adoperati per mostrare gli elementi virtuali nel mondo reale, così facendo l'utilizzatore può interagire con essi senza alcun altro device.

1.3.2 Realtà Virtuale

Per VR s'intende una tecnologia in grado di catapultare il suo utilizzatore in un mondo completamente virtuale escludendo quello reale. Gli ambienti creati possono essere: riproduzioni di ambienti reali e/o ambienti completamente immaginari. L'esperienza utente in un contesto VR è multisensoriale, infatti in essa sono inclusi: suoni, immagini a 360° e interazioni con gli oggetti virtuali (grazie a dispositivi di motion capture). Come nel caso della MR, tutto ciò che l'utente vede rispetta angolazioni e prospettive. I dispositivi utilizzati sono: HMD (solitamente collegato a una console o computer) e particolari tipologie di tute. L'utilizzo esclusivo di questi specifici dispositivi è dovuto al fatto che l'utente debba essere completamente calato in un mondo diverso da quello esterno, cosa che si può ottenere solo con questa tipologia di devices.

1.3.3 Differenze tra AR, MR e VR

Nonostante a primo acchito possano sembrare tecnologie molto simili tra loro, possiamo subito individuare una differenza fondamentale della VR rispetto a MR e AR. La VR è una tecnologia che proietta il suo utilizzatore in un mondo completamente virtuale, estraniandolo, di fatto, da quello reale. Contrariamente AR e MR arricchiscono il mondo circostante con oggetti virtuali senza mai escluderlo. Quindi ciò che ci chiediamo è: qual è la differenza tra AR e MR? Spesso questi due

termini sono impropriamente usati come sinonimi, tuttavia vi è una sottile ma sostanziale differenza. Quando parliamo di AR ci riferiamo all'inserimento di testi o immagini nel mondo reale, mentre per MR s'intende la creazione di un ambiente in cui si può interagire con gli oggetti virtuali come se fossero reali. Quindi possiamo affermare che, in un certo senso, la MR è il punto d'incontro tra la realtà virtuale e la realtà aumenta.

1.4 Utilizzi della Realtà Aumentata

Oggigiorno l'AR ci circonda ed è usata per i più disparati settori, fra questi possiamo trovare: marketing, intrattenimento, turistico, culturale, manutenzione, moda, didattico, videogiochi e così via. Di seguito faremo qualche esempio di applicazioni AR.

1.4.1 Settore didattico

Carmen Juan, Francesca Beatrice e Juan Cano hanno realizzato uno dei primi sistemi di realtà aumentata con scopi didattici. Il sistema aveva come obiettivo l'apprendimento delle strutture anatomiche del corpo umano. L'utente poteva "aprire" l'addome di un corpo umano virtuale utilizzando le proprie mani, al fine di vedere le aree dello stomaco e dell'intestino. Il sistema prevedeva l'utilizzo di una telecamera

collegata a un HMD. Le immagini virtuali proiettate nel mondo reale erano situate su di uno specifico marker [33].

1.4.2 Settore manutenzione

Steven J. Henderson e Steven K. Feiner hanno realizzato un sistema di realtà aumentata per migliorare e velocizzare le procedure di montaggio e smontaggio di un motore a turboelica Rolls-Royce Dart 510. L'utente è guidato step by step nel montaggio/smontaggio del motore, grazie ai suggerimenti che visualizza sul proprio HMD, che risulta essere necessario per il funzionamento del sistema. Tutti i partecipanti hanno mostrato una maggiore velocità e correttezza nel montaggio/smontaggio del motore [37].

1.4.3 Settore marketing

La casa automobilistica MINI, creò una pubblicità AR presente su una serie di riviste tedesche. Il lettore doveva solamente andare sul sito ufficiale MINI, mostrare l'annuncio in web-cam e sullo schermo veniva visualizzata un'automobilina in 3D [45]. Quindi il sistema AR non faceva altro che riconoscere il marker (la pagina della rivista) e piazzare su di esso il modellino virtuale 3D della macchinina.

Capitolo 2

Architettura di un sistema AR

Nella trattazione di questo capitolo sarà definito: cosa s'intende per architettura motivandone l'importanza, gli attributi di qualità richiesti a un sistema AR, macro-blocchi concettuali che lo compongono, quale può essere considerata una sua architettura di base e quali componenti ne fanno parte.

2.1 Architettura Software

Nel campo dell'ingegneria del software esistono moltissime definizioni diverse di architettura software, presenti sul sito web del Software Engineering Institute [4], fra esse troviamo:

- *"Software Architecture = Elements, Form, Rationale"*(Perry e Wolf) [57];
- *"The fundamental organization of a system embodied in its components, their relations to each other, and to the environment, and the principles guiding its design and evolution"*(IEEE 1471) [43];
- *"The set of principal design decisions governing a system"* (Taylor, Medvidovic, and Dashofy) [36].

Si può quindi affermare che l'architettura software è composta da elementi, dalle loro connessioni e da altri aspetti come: proprietà, vincoli, requisiti e le esigenze delle parti interessate. Fatta chiarezza circa cosa sia un'architettura software ora bisogna capire perché è importante. Spesso si pensa che l'architettura abbia come unico scopo il corretto funzionamento del sistema. Ciò è vero solo in parte, poiché un sistema potrebbe tranquillamente funzionare anche con un'architettura non ben definita. Il vero scopo dell'architettura è supportare il ciclo di vita del sistema in modo tale da renderlo facile da sviluppare, manutenere, riusare e capire, riducendo così costi e tempi di sviluppo.

2.2 Attributi di qualità

Come accennato nel paragrafo precedente, a un sistema software sono richiesti un insieme di attributi di qualità per supportarne il ciclo

di vita e il funzionamento. Ciò su cui bisogna fare chiarezza è quali tipologie di attributi sono ritenuti importanti in un sistema AR. Quanto detto è affrontato nel paper *"Study on Software Architectures for Augmented Reality Systems"* [31], in cui gli attributi sono classificati in due categorie: frequenti e non frequenti, che a loro volta si suddividono in attributi: run-time e non run-time. La differenziazione, tra queste categorie, è stata effettuata somministrando un questionario ai realizzatori di sistemi AR applicati a diversi domini. Il questionario, visibile nell'appendice del suddetto paper, contiene una serie di attributi, per ognuno è richiesta la scelta tra le descrizioni proposte riguardanti la sua severità in quella determinata applicazione. La classificazione degli attributi, discendente dall'analisi delle risposte, è la seguente:

- Alcuni tra i frequenti sono: latenza di tracking e rendering, funzionamento outdoor e modificabilità;
- Alcuni tra i non frequenti sono: libertà di movimento, riconfigurabilità a run-time e carico imposto alla CPU.

Per la classificazione completa si rimanda al paper [31].

2.3 Struttura a macro-blocchi di un sistema AR

Prima di introdurre l'architettura software di base di una sistema AR è bene farsi un'idea generale della sua composizione, considerando: utilizzatore, hardware e software. In questo modo si avranno più chiare le problematiche da affrontare per la sua progettazione e realizzazione. Si consideri la seguente figura proposta nel libro *"Spatial Augmented Reality"* di Bember e Raskar [50], essa è composta da quattro livelli, ognuno contenente uno o più blocchi che saranno oggetto di spiegazione.

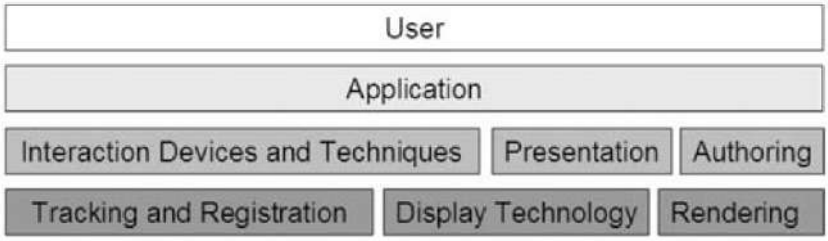


Figura 2.1: Figura a macro-blocchi [50]

Partendo dall'altro troviamo:

- *User*: l'utilizzatore finale del sistema;
- *Application*: fornisce l'interfaccia finale che permetterà l'utilizzo del sistema da parte dell'utente;
- *Interaction Devices and Techniques*: le tecnologie hardware su cui deployare le applicazioni AR;

- *Presentation*: permette agli oggetti virtuali di apparire realistici mediante l'utilizzo dei dati forniti dal rendering;
- *Authoring*: piattaforme di sviluppo di applicazioni AR;
- *Tracking and Registration*: permette di registrare la posizione degli oggetti reali e virtuali. Le caratteristiche da cui non può prescindere sono: velocità e precisione. La precisione è ottenuta analizzando i dati della scena reale osservata mediante, ad esempio, l'utilizzo di telecamere. Quest'ultime: se utilizzate come *marker-based*, verificheranno la presenza di marker, ossia oggetti predefiniti da usare come punti di riferimento; se utilizzate come *markerless*, verificheranno il soddisfacimento di certe proprietà della scena osservata (ad esempio se sono stati effettuati determinati gesti);
- *Display Technology*: permette di collocare gli oggetti virtuali sui riferimenti individuati dal tracking;
- *Rendering*: permette l'ottenimento dei dati che consentirà agli oggetti virtuali di apparire realistici; Ciò è ottenuto utilizzando il calcolo della prospettiva, delle immagini a 360°, dei colori, delle luci e delle ombreggiature.

Escludendo i blocchi: User, Interaction Devices and Techniques e Authoring, i restanti sono le componenti software da cui un'applica-

zione AR non può prescindere e come tali saranno il fulcro attorno al quale costruire l'architettura software.

2.4 Architettura Software generale di un sistema AR

I sistemi di Realtà Aumentata, nonostante delle differenze dovute agli ambiti di applicazione, condividono un'architettura base. In essa si possono trovare i blocchi fondamentali, descritti nel paragrafo precedente, che qui saranno indicati con il termine più appropriato di componenti. La loro presenza è necessaria per conferire al sistema tutte le proprietà che contraddistinguono la Realtà Aumentata. Proprio questa comunanza delle funzionalità principali permette l'ottenimento dell'architettura di riferimento [32].

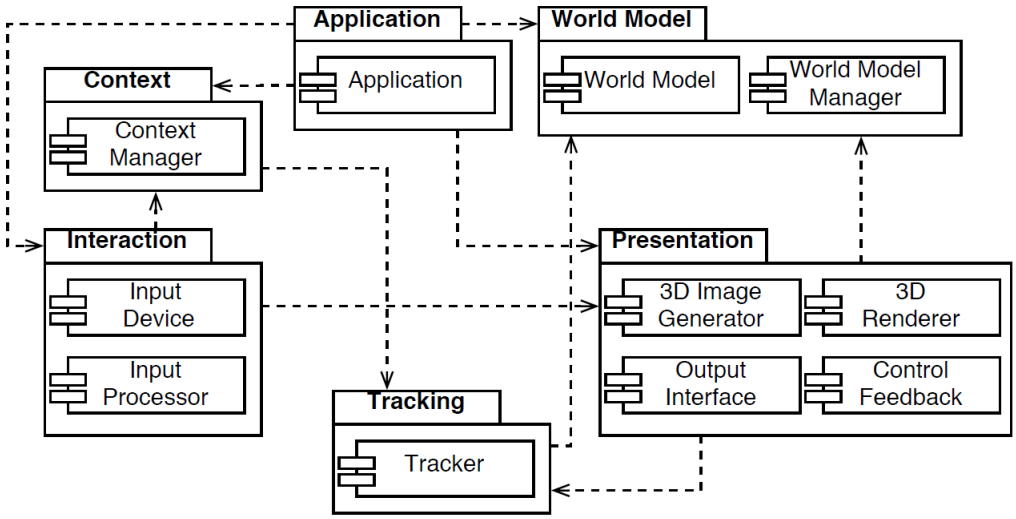


Figura 2.2: Architettura di base [32]

La figura 2.2 mostra il Package Diagram dell'architettura di riferimento, che può essere decomposta in sei sottosistemi principali, ognuno rappresentato con il proprio package. All'interno di ciascuno di essi sono presenti uno o più componenti che a loro volta potrebbero essere decomposti [32]. Di seguito si analizza ognuno dei sei sotto-sistemi:

- *Application*: corrispettivo dell'omonimo blocco, ingloba tutto il codice necessario per una specifica tipologia di applicazione;
- *Interaction*: recepisce ed elabora gli input provenienti dall'utente;
- *Presentation*: incorpora i blocchi *Presentation*, *Rendering* e *Display Technology*, mostra l'output all'utente gestendo gli oggetti virtuali;
- *Tracking*: corrispettivo dell'omonimo blocco, responsabile della registrazione della posizione di oggetti reali e virtuali, di cui dovrà fornire i dati. E' importante sottolineare che dovrà essere eseguito sempre e parallelamente ad altre attività del sistema;
- *Context*: raccoglie i diversi tipi di dati di contesto tra cui le preferenze e le attività dell'utente;
- *World Model*: raccoglie le informazioni legate al mondo reale in cui si muove l'utente.

Quanto emerso dallo studio dei singoli sotto-sistemi ci permette di riscontrare delle similitudini tra l'architettura generale risultante e quella già ben consolidata *MVC*. *MVC* è l'acronimo di:

- *Model*, responsabile di mantenere una rappresentazione dei dati dell'applicazione;
- *View*, l'insieme degli elementi visuali presentati all'utente;
- *Controller*, coordina i cambiamenti al Model e alla View in base alla logica di business.

Volendo sovrapporre le due architetture, il sotto-sistema *Interaction* risulterebbe il corrispettivo di *Controller* e *Presentation* il corrispettivo di *View* [32]. Si può quindi concludere che l'architettura generale di un sistema AR risulta molto simile all'MVC.

2.4.1 Class Diagram generale dei sotto-sistemi principali

Di seguito è proposto il Class Diagram dell'architettura di riferimento, che sarà particolarizzato per ognuno dei sei sotto-sistemi che lo compongono (citati nel paragrafo precedente). L'obiettivo è mostrare, mediante decomposition refinement, quali sono le classi principali interne a ciascun sotto-sistema.

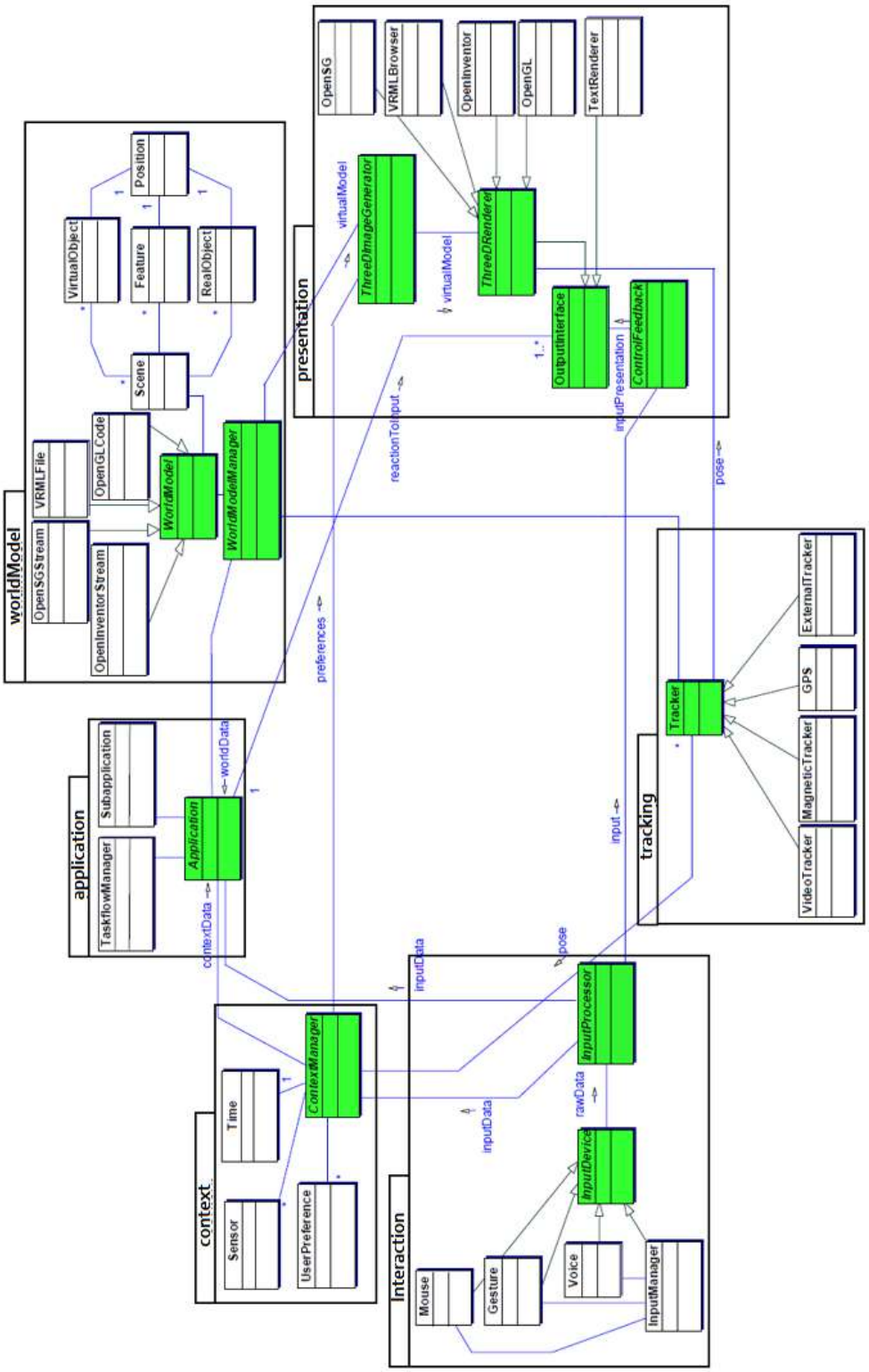


Figura 2.3: Class Diagram architettura di base

Application

Il sotto-sistema Application presenta il codice relativo alla specifica applicazione fornendo l'interfaccia per l'utente ed è responsabile del bootstrap. L'applicazione può essere a sua volta composta da molte altre applicazioni secondarie e componenti. Questo tipo di sotto-sistema non è specifico per la Realtà Aumentata. Nel Class Diagram Application è una classe astratta e come tale può essere implementata in diversi modi[31].

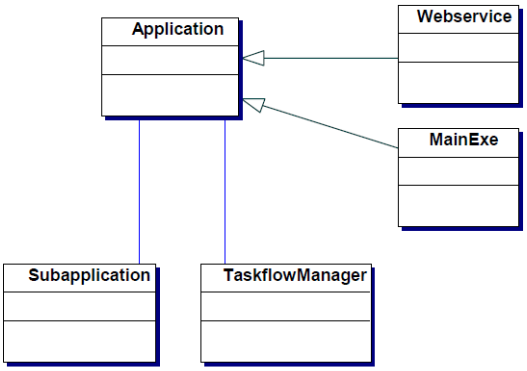


Figura 2.4: Class Diagram Application [31]

Tracking

Il sotto-sistema Tracking è specifico per la Realtà Aumentata, il suo compito è tracciare la posizione degli oggetti reali e virtuali analizzando i dati ottenuti dal WorldModel. Dopo aver ottenuto la posizione, questa viene inoltrata al componente *ThreeDReneder* e al *ContextManager* [31].

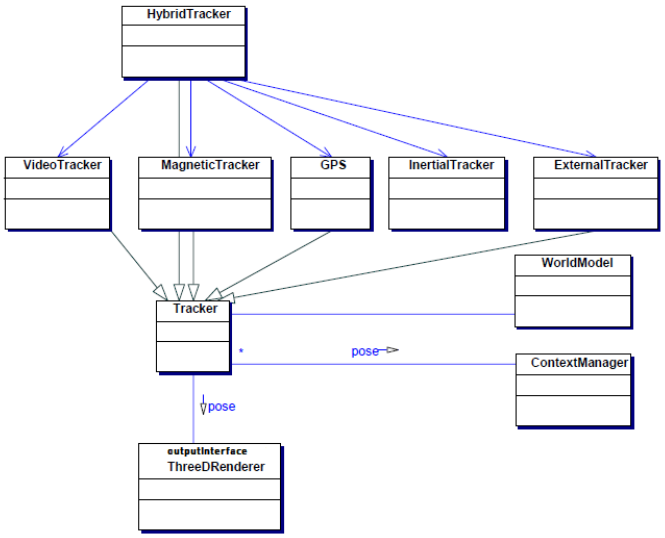


Figura 2.5: Class Diagram Tracking [31]

Interaction

Il sotto-sistema Interaction recepisce qualsiasi input dell'utente. I dati sono creati dalle implementazioni della classe astratta *InputDevice* e sono del tipo: touch, gesti, voce ... o una loro combinazione creata dalla classe *InputManager*. I dati ricevuti sono inoltrati all'*InputProcessor* che li interpreta utilizzando le classi *DataAnalysis* e *ModalityFusion*. Il risultato ottenuto è, a sua volta, inoltrato al *ContextManager*, all'*Application* e al *ControlFeedbaback*. Quest'ultimo, essendo parte del sotto-sistema Presentation, visualizza l'input riconosciuto [31].

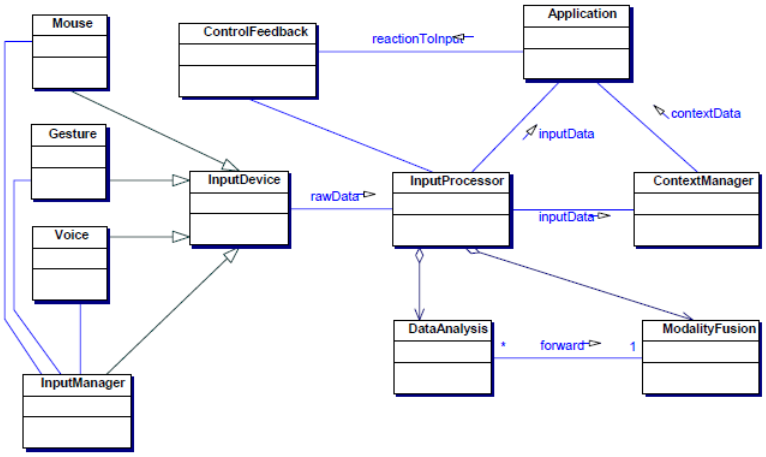


Figura 2.6: Class Diagram Interaction [31]

Presentation

Il sotto-sistema Presentation gestisce la grafica degli oggetti virtuali. La classe *ThreeDRender* risulta essere il core dell'intero sotto-sistema poiché su di essa ricade l'onere di generare gli oggetti virtuali. Essi devono rispettare prospettiva, posizione e angolazione rispetto ai dati forniti dal Tracker. La classe *ThreeDImageGenerator* genera le immagini adattando gli oggetti virtuali di WorldModel al contesto [31].

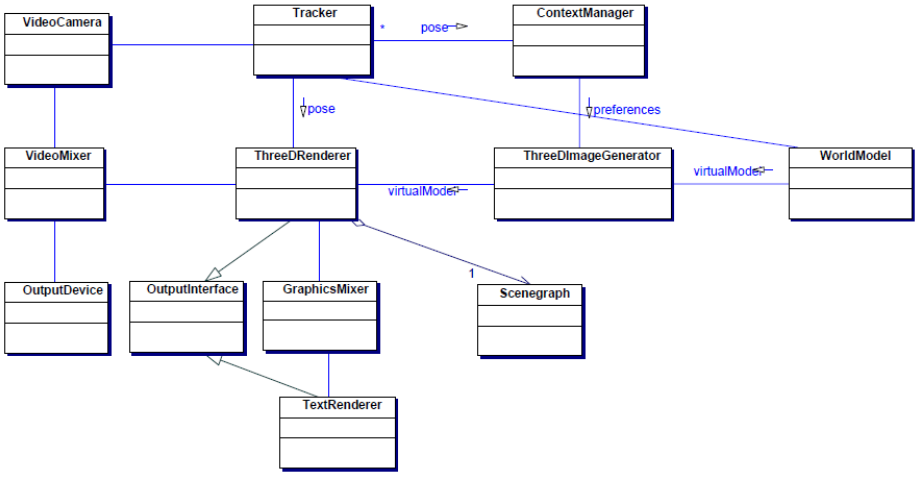


Figura 2.7: Class Diagram Presentation [31]

Context

Il sotto-sistema Context tiene traccia dei dati di contesto e delle preferenze dell'utente, che variano da dominio a dominio. *ContextManager* raccoglie i diversi tipi di informazione sul contesto prima citate e le rende accessibili per gli altri componenti. *ContextProcessor*, può utilizzare le informazioni di contesto, elaborarle e generarne di nuove [31].

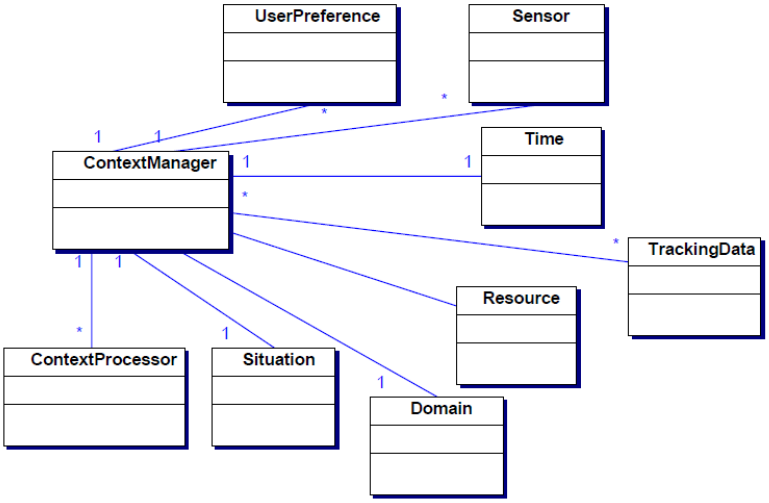


Figura 2.8: Class Diagram Context [31]

World Model

Il sotto-sistema World Model raccoglie le informazioni legate al mondo reale in cui si muove l'utente. Il *WorldModelManager* gestisce gli accessi al WorldModel, che è composto da scene a loro volta costituite da: oggetti virtuali, informazioni degli oggetti reali e informazioni per i tracker. Come si può intuire l'informazione principale è la posizione di

questi oggetti calcolata in base alle coordinate del mondo circostante [31].

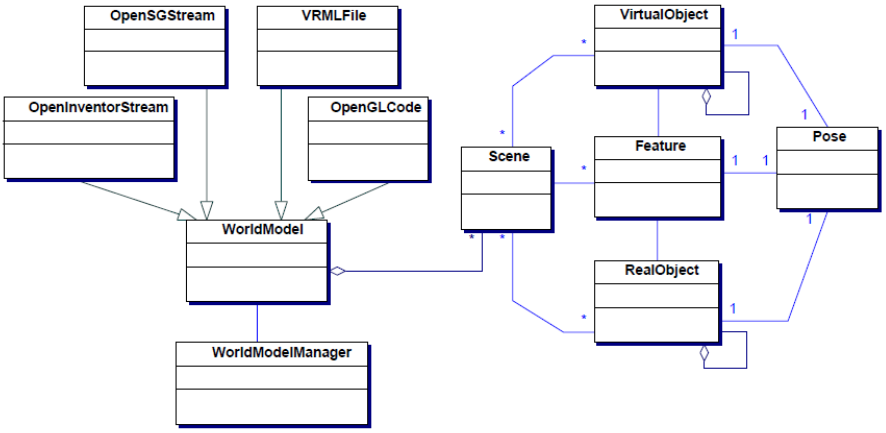


Figura 2.9: Class Diagram World Model [31]

2.4.2 Sequence Diagram delle classi core

Per meglio comprendere il funzionamento generale relativo all'interazione delle varie classi, si è deciso di ricorre al Sequence Diagram. Esso permetterà di mettere in luce come gli oggetti delle varie classi interagiscono per portare alla creazione della scena di realtà aumentata. In particolare si fa presente che l'architettura generale proposta non include metodi specifici. Pertanto i messaggi scambiati, dai vari oggetti, non rappresentano delle vere e proprie invocazione di funzioni con i relativi parametri, ma piuttosto indicano le informazioni generali scambiate dai vari oggetti. Per quanto riguarda la notazione di questi ultimi, è stata definita nel seguente modo: NomeClasse:NomePackage, per questioni di chiarezza. Il Sequence Diagram della figura 2.10 presenta esclusivamente le classi core di ogni package. Dal diagramma

si evince che l'interazione dell'utente con il sistema è opzionale, ciò significa che la scena di realtà aumentata, appena avviata l'applicazione, sarà comunque creata e mostrata. Ciò avviene perché è presente un altro "attore" che interagisce con il sistema, ossia il mondo reale. La scelta di trattarlo come tale è stata fatta poiché, come l'utente, ogni variazione del mondo reale risulta essere una possibile interazione per l'applicazione AR e per tale ragione è necessario che sia valutata. Più precisamente il diagramma mostra che le caratteristiche del mondo reale, recepite da dispositivi quali ad esempio: sensori e telecamere, sono salvate all'interno di "WorldModel". Tale valutazione dell'ambiente circostante è effettuata costantemente e in contemporanea con eventuali input dell'utente. Ciò sottolinea quanto il mondo reale, in cui è calata l'applicazione AR, giochi un ruolo di fondamentale importanza. L'interazione dell'utente con il sistema AR avviene mediante devices (smartphone, tablet, HMD ...), che producono informazioni salvate da "InputDevice" e successivamente inoltrate alla "InputProcessor", che le elabora e le invia: ad "Application" per la logica applicativa variante in base agli input dell'utente, a "Context" per il salvataggio delle preferenze dell'utente e a "ControlFeedBack" per la presentazione grafica degli input. Similmente, le informazioni salvate in "WorldModel" sono inoltrate al "WorldModelManager" che le elabora e le inoltra: a "Tracker" per definirne la posizione, a "ThreeDImageGenerator" per l'adattamento degli oggetti virtuali in funzione

di quelli reali e ad "Application" per la logica applicativa variante in base alle condizioni del mondo esterno. Per quanto riguarda "Tracker", come anticipato, è responsabile della definizione della posizione degli oggetti reali (in particolare dei marker, utilizzati come riferimenti di posizionamento degli oggetti virtuali) e del conseguente inoltro al "ThreeDRender" e al "ContextManager". Quest'ultimo è responsabile del mantenimento delle informazioni relative al contesto in cui si trova l'applicazione e delle preferenze dell'utente che interagisce con il sistema. Tali informazioni sono utili per la logica applicativa e per tale ragione sono inoltrate ad "Application". Questi, come sottolineato precedentemente, è responsabile della logica applicativa dell'applicazione AR e avendo a disposizione i dati: di contesto, del mondo reale ed eventualmente dell'interazione dell'utente, elabora i dati di reazioni agli input dei due attori e successivamente li trasmette alla "OutputInterface". Prima di illustrare le funzionalità di quest'ultima è necessaria una previa spiegazione di "ThreeDImageGenerator", "ThreeDRender" e "ControlFeedBack". "ControlFeedBack", in caso di interazione dell'utente, riceve le elaborazioni di tali dati, li elabora a sua volta per la presentazione grafica e li invia alla "OutputInterface". "ThreeDImageGenerator", avendo a disposizione le informazioni relative al mondo reale e quelle di contesto, ha l'onere di elaborarle al fine di fornire i dati per il rendering degli oggetti virtuali a "ThreeDRender". Quest'ultimo, avendo ricevuto anche i dati relativi al posizionamento degli

oggetti reali, è atto a creare i dati degli oggetti virtuali per la scena di realtà aumentata. Tali dati saranno inoltrati alla "OutpuInterface" che li utilizzerà, comunemente ai dati ricevuti dal "ControlFeedBack", per mostrare all'utente la scena di realtà aumentata.

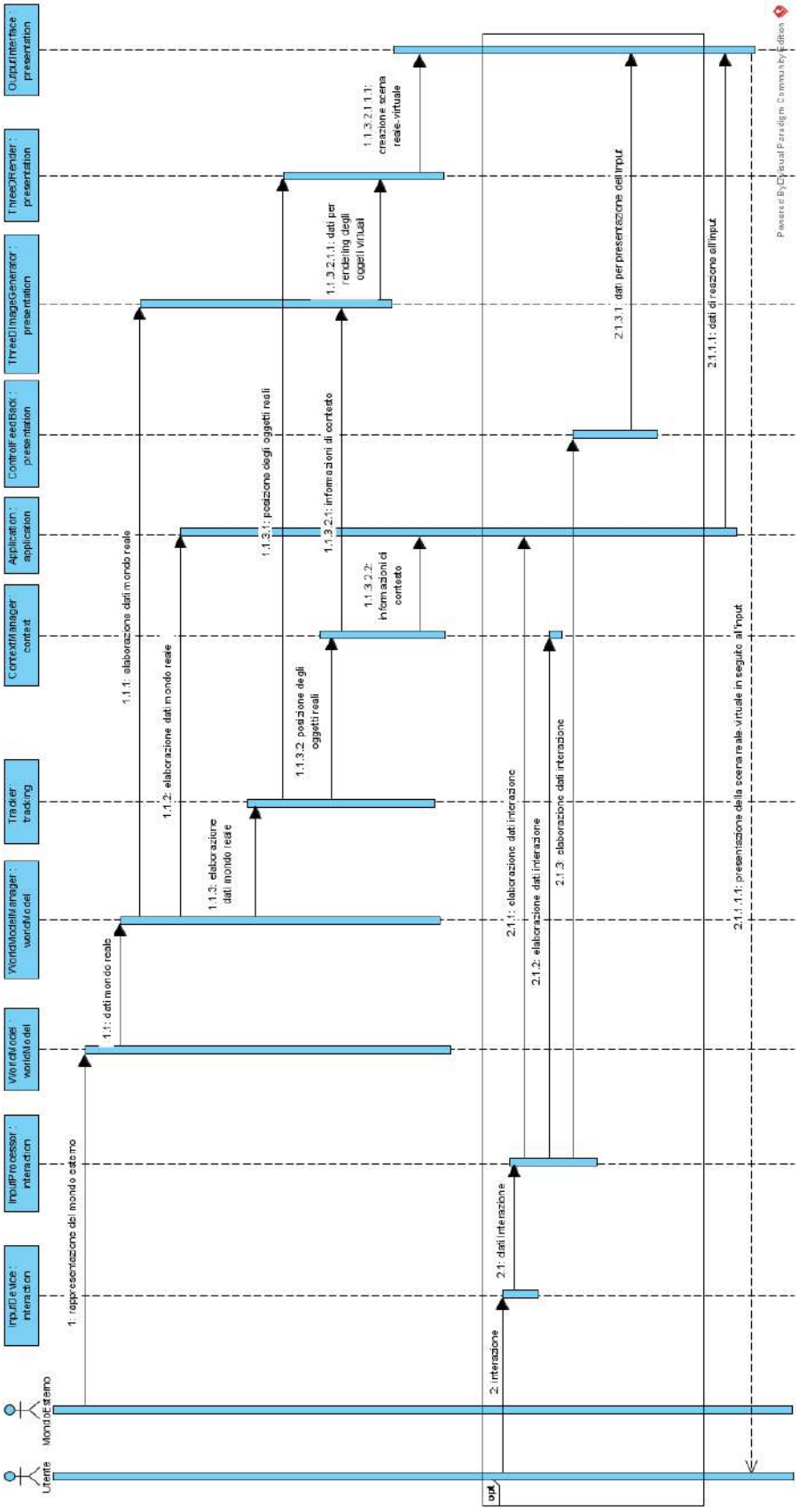


Figura 2.10: Sequence Diagram classi core

2.5 Esempio di deploy

In ultima istanza si vuole mostrare come sia deployato un sistema di realtà aumentata sulle componenti hardware. Per componenti hardware s'intende: il dispositivo che permette la riproduzione di contenuti virtuali congiuntamente a quelli reali e, eventualmente, un server da cui attingere i dati per la creazione della scena di realtà aumentata. Tuttavia non sempre è necessaria la presenza di un server in grado di fornire i dati, infatti si potrebbe decidere di salvarli tutti all'interno della stessa applicazione evitandone l'utilizzo. Questo tipo di soluzione però è strettamente legata alle dimensioni dell'applicazione AR, poiché è realizzabile quasi esclusivamente per applicazioni medio-piccole come semplici giochi o poco più. Per mostrare un esempio di applicazione AR deployata, si presenta un progetto del 2016 realizzato dall'università colombiana "Institución Universitaria Salazar y Herrera". Il progetto è per dispositivi mobili (smartphone) e ha scopi didattici, infatti l'applicazione realizzata mira a confrontare due tipologie di approcci di realtà aumentata per lo studio di alcuni concetti del corso di "Fundamentals of Electronics", che si tiene nella stessa università. Per ulteriori chiarimenti si rimanda a [49]. Il diagramma atto a mostrare come le varie componenti del progetto risultino essere deployate sui dispositivi hardware è il Deployment Diagram. Di seguito è mostrato quello relativo al suddetto progetto.

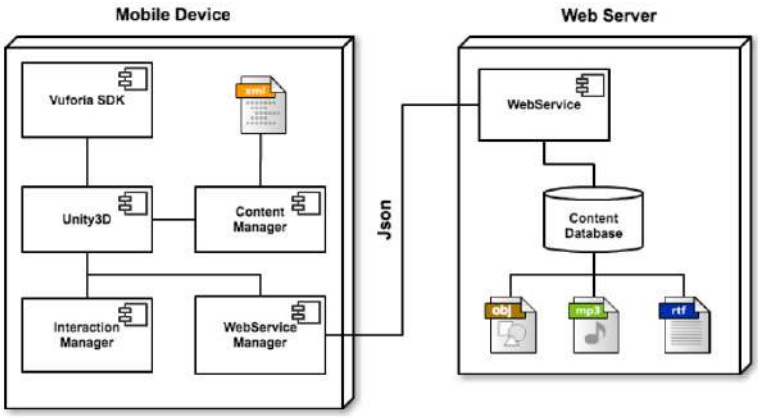


Figura 2.11: Deployment Diagram [49]

Il Deployment Diagram mostra un'architettura di tipo two-tier client-server, rispettivamente il dispositivo mobile e il web server. Nel client sono presenti i seguenti componenti:

- Unity e Vuforia, implementano le classi dell'architettura generale mostrata nella sezione 2.2 e hanno l'onere di creare la scena di realtà aumentata. Entrambi gli strumenti saranno trattati più nel dettaglio nel capitolo 3;
- "Content Manager", gestisce le diverse tipologie di contenuti mediante il formato XML. Tali contenuti sono utilizzati da Unity e Vuforia per gli scopi esplicitati al punto precedente;
- "Interaction Manager", gestisce le diverse modalità di interazione con l'applicazione;
- "WebService Manager", consente di comunicare con il web server mediante protocollo JSON.

Per quanto riguarda il server, i componenti in esso presenti sono i seguenti:

- "WebService": riceve le richieste JSON, preleva le risorse richieste dal database e le restituisce al client;
- "Content Database": database per l'archiviazione delle risorse, in particolare file .obj per modelli 3D, file .mp3 per l'audio e file .rtf per le descrizioni testuali.

Capitolo 3

Tools per la realizzazione di un sistema AR

In questo capitolo saranno introdotti gli strumenti necessari alla realizzazione di un sistema AR, effettuando una breve panoramica riguardante gli SDK (Software Development Kit) e gli Engine in cui è possibile importarli. Successivamente saranno descritti più nel Dettaglio Vuforia e Unity, rispettivamente SDK ed Engine, scelti come tecnologie target.

3.1 SDK per la Realtà Aumentata

Per SDK (Software Development Kit) s'intende un "pacchetto" di tools che rende possibile e facilita la programmazione di software per una specifica piattaforma o applicazione. Un SDK contiene le API (Appli-

cation Programming Interface), invocabili dagli sviluppatori, che incapsulano funzionalità complesse. Questo è il caso degli SDK di realtà aumentata, che mettono a disposizione le funzionalità basilari dell'AR come rendering e tracking [40]. Ad oggi esistono diversi SDK per la realtà aumentata, di seguito sarà effettuata una panoramica di alcuni di essi, che permettono l'integrazione con i diversi Engine e lo sviluppo di applicazioni mobile.

3.1.1 ARKit

È l'SDK in grado di fornire tutte le funzionalità di base (come mostrato nella documentazione ufficiale [5]) necessarie allo sviluppo di applicazioni di realtà aumentata per iOS, il sistema operativo degli smartphone e tablet Apple. Tra le molteplici funzionalità, ARkit consente di esaminare la scena, proveniente dalla telecamera, riuscendo a distinguere superfici piane e a misurare l'illuminazione. In questo modo sulle superfici piane possono essere posizionati oggetti virtuali che saranno illuminati con una luce che rispetterà le condizioni di luminosità dell'ambiente in cui si trova [42].

3.1.2 ARCore

È l'SDK, per lo sviluppo di applicazioni AR in Android che, similmente ad ARKit, fornisce le funzionalità di base necessarie (come mostrato nella documentazione ufficiale [6]). Negli ultimi rilasci è stata migliora-

ta la funzionalità per il calcolo della profondità e dell'occlusione (utile per il calcolo della presenza di oggetti virtuali davanti o dietro quelli reali), permettendo una percezione ancora più realistica degli oggetti virtuali. Oltre a ciò, sono state inserite delle API per lo sviluppo di applicazioni iOS, tuttavia sono ancora in numero abbastanza esiguo e quindi si può confermare che ARCore resta un SDK utilizzato in Android.

3.1.3 Wikitude

È un ulteriore SDK per la realtà aumentata, di cui è possibile trovare la lista completa delle sue funzionalità di base nella documentazione ufficiale [7]. Differentemente rispetto agli SDK precedentemente descritti, Wikitude permette lo sviluppo di applicazioni multipiattaforma (iOS e Android). È disponibile in versione commerciale e in versione di prova con qualche limitazione [40].

3.1.4 ARToolKit

Anch'esso permette lo sviluppo di applicazioni di realtà aumentata, la lista delle sue funzionalità è presente nella documentazione [8]. Diversamente dagli SDK precedenti, nacque per lo sviluppo di applicazioni desktop, solo successivamente sono state sviluppate delle estensioni che supportano anche piattaforme mobile [40]. Di conseguenza,

come si può intuire, le applicazioni sviluppate con ARTololKit sono multipiattaforma.

3.1.5 Vuforia

Permette lo sviluppo di applicazioni AR, la lista delle sue funzionalità è consultabile al link della documentazione ufficiale [9]. Le applicazioni sviluppate con Vuforia sono multipiattaforma grazie al supporto per ARCore e ARKit. Tra le funzionalità spicca la possibilità di poter scannerizzare un oggetto di piccole dimensioni [42] per poi utilizzarlo come marker 3D, sfruttandolo come punto di riferimento per piazzare gli oggetti virtuali.

3.1.6 Confronto

Fatta la panoramica di alcuni dei maggiormente utilizzati SDK per la realtà aumentata, è necessario sceglierne uno come target. Di seguito saranno effettuati dei confronti, tra gli SDK precedentemente descritti, per valutarne le caratteristiche. I primi SDK ad essere estromessi dalla scelta sono ARCore e ARKit in quanto, rispetto agli altri, non sono multi piattaforma poichè supportano il solo sviluppo di applicazioni AR rispettivamente per Android e iOS. Dei rimanenti tre, ARToolKit non risulta essere nativamente sviluppato per dispositivi mobili, ma necessità di estensioni per il supporto di piattaforme mobile. La scelta, quindi, deve essere effettuata tra Vuforia e Wikitude. Quest'ultimo

presenta delle limitazione nella sua versione free e per questa ragione l'SDK su cui ricade la scelta è Vuforia.

3.2 Engine

Per Engine s'intende l'insieme degli strumenti comuni ad applicazioni prevalentemente grafiche. Tra questi componenti è possibile trovare: animazione, rilevamento di collisioni, motore fisico e intelligenza artificiale (AI) [29]. Solitamente tutti questi strumenti sono forniti in un IDE (Integrated Development Environment) per far sì che siano utilizzati in modo semplice e veloce. Proprio questa è la potenza degli Engine, ossia fornire una piattaforma riutilizzabile per le funzionalità di base delle suddette applicazioni ottimizzando, non solo, il lavoro degli sviluppatori ma anche costi e tempi di sviluppo. Altro punto di forza degli Engine è la capacità di rendere l'applicazione sviluppata multipiattaforma, ciò significa che è possibile esportarla in diverse piattaforme (smartphone Android e iOS, HMD, PlayStation, Xbox ...) senza la necessità di alcuna modifica.

3.2.1 Unity e Unreal

Entrambi sono Engine che presentano le caratteristiche citate precedentemente, sono i più utilizzati per lo sviluppo di applicazioni 2D e 3D. Infatti tra i vari prodotti sviluppati da Unity e da Unreal so-

no presenti non solo videogame ma anche cortometraggi animati dalla grafica realistica. Proprio per questo realismo degli oggetti virtuali, già da qualche lustro sono utilizzati anche come piattaforme per lo sviluppo di applicazioni di realtà aumenta, mista e virtuale.

3.2.2 Confronto

Una volta introdotti i due principali Engine è necessario sceglierne uno come tecnologia target. La dicotomia Unity e Unreal è un must che ci accompagna da qualche anno e vede i sostenitori, dell'uno e dell'altro, praticamente bilanciati. Per questo motivo la scelta è stata effettuata semplicemente considerando la numerosità della community su reddit poiché, in caso di problemi, sicuramente sarà trovato un post nel forum o una pagina della documentazione in grado di risolverlo. I numeri delle community dei due Engine sono:

- Unity3D: 229k;
- Unreal Engine: 106k.

Quindi, come si può facilmente intuire, l'Engine scelto come tecnologia target è Unity.

3.3 Vuforia

Come detto nel paragrafo 3.1.5, Vuforia è l'SDK scelto in grado di combinare il mondo reale con gli oggetti virtuali. Vuforia è stato lan-

ciato nel 2010 da Qualcomm e, successivamente, nel 2015 acquistato da PTC, multinazionale specializzata in IoT (Internet of Things) e realtà aumentata. Vuforia supporta una grande molteplicità di dispositivi, come ad esempio Microsoft HoloLens, Epson Moverio, smartphone, tablet, e PC. Nella figura 3.1 si mostra la sua architettura, da essa è possibile evincere parte dei componenti presentati nell'architettura di riferimento 2.4. In particolare le componenti in essa presenti sono: Tracker e Presentation. Nella fattispecie: Tracker è utilizzato da Vuforia per il riconoscimento dei marker e per il tracciamento della posizione degli oggetti reali e virtuali; Presentation ha il compito di mostrare la scena di realtà aumentata creata grazie al rendering degli oggetti virtuali presentati insieme a quelli reali.

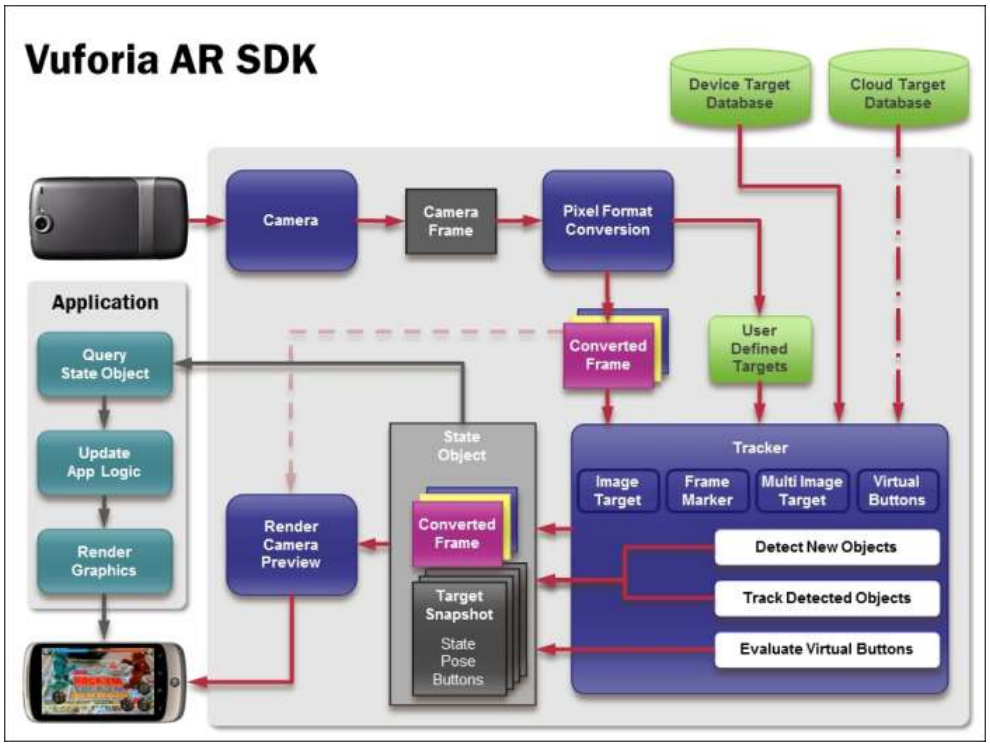


Figura 3.1: Architettura Vuforia [41]

Come si può notare dall'architettura, il lavoro dello sviluppatore è molto semplificato, questi dovrà semplicemente definire la logica dell'applicazione senza preoccuparsi di come sarà realizzato rendering e tracking. Solitamente le applicazioni che fanno uso di Vuforia sfruttano la funzionalità di riconoscimento dei marker, utilizzati come punto di riferimento per il piazzamento degli oggetti virtuali. Riguardo i Marker, Vuforia permette di riconoscerne più di uno contemporaneamente e di scegliere immagini custom. Quando si scelgono immagini custom è possibile importarle nell'applicazione o utilizzare il "Cloud Recognition". Quest'ultimo permette di mantenere i target in cloud senza doverli integrare nell'applicazione. Riguardo le immagini custom, devono essere caricate nel "Target Management System", ossia lo strumento online di Vuforia per creare marker a partire da immagini custom [34]. Da esse vengono estratte delle caratteristiche naturali, che saranno utilizzate per rilevare real-time il marker durante l'esecuzione dell'applicazione. Il numero di caratteristiche naturali estratte dall'immagine definisce uno star-rate da 1 a 5. Maggiore è il numero di stelle più facilmente è possibile riconoscere il marker. Le caratteristiche naturali rilevate da Vuforia sono nitidezza dei dettagli, gli angoli e la texture [27].

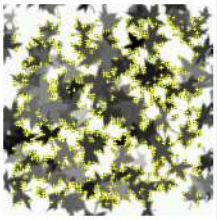
Information	Object	Analysis	Star Rate
Image with few features			★☆☆☆☆
Image with many features			★★★★★

Figura 3.2: Star-rate [27]

Le API messe a disposizione sono in linguaggio C++, Java e .NET, grazie ad esse è possibile creare applicazioni in linguaggio nativo o integrarlo con un Engine. Per poter utilizzare Vuforia è necessario creare, gratuitamente, un account sul sito ufficiale per gli sviluppatori [10].

3.4 Unity

Unity è un Engine multipiattaforma sviluppato da Unity Technologies, un'azienda fondata in Danimarca che si occupa principalmente di sviluppo video-game 2D e 3D per console, PC e smartphone [47].

La prima versione dell'Engine, Unity 1.0, è stata rilasciata nel 2005. Come accennato nel paragrafo 3.2, Unity presenta una serie di componenti fondamentali per lo sviluppo di giochi e applicazioni, tra essi troviamo:

- Intelligenza Artificiale: unisce scenari pre-calcolati a comportamenti generati in base alle condizioni del particolare momento di gioco, al fine di dare l'illusione di comportamenti intelligenti solitamente attuati da personaggi non giocanti;
- Motore fisico: associa le leggi della fisica al mondo virtuale creato, fornendo un'esperienza di gioco realistica. In particolare la texture e i materiali scelti per realizzare un oggetto virtuale contribuiranno a determinare le caratteristiche fisiche del suddetto oggetto;
- Input: ha l'onere di acquisire i comandi provenienti da joystick e tastiere, richiedendo una codifica minima per associare loro gli script di movimento;
- Rendering multimediale: generano gli elementi grafici e riproducono eventuali tracce audio. In merito agli elementi grafici per la loro generazione Unity utilizza la librerie OpenGL, OpenGL ES e WebGL;

- Networking: implementa routine di comunicazione, per giochi multi-player e su server, con la prerogativa fondamentale della velocità.

Volendo mappare i componenti presentati nell'architettura di riferimento 2.4, in Unity troviamo: Application responsabile della logica applicativa, Interaction responsabile delle interazioni dell'utente con gli oggetti virtuali, WorldModel responsabile delle informazioni relative al mondo esterno e Context responsabile delle informazioni di contesto e dell'utente. Tali componenti sono presenti in Unity poiché queste informazioni e azioni sono gestite mediante script, infatti Unity include un proprio IDE e un framework di scripting orientato agli oggetti in linguaggio C#. Per quanto riguarda la programmazione, gli oggetti utilizzati da Unity sono i "GameObject", a cui non sempre corrisponde una rappresentazione grafica. A ogni GameObject può essere associato uno script per definirne il comportamento. Gli script sono estensioni della classe base "MonoBehaviour" in cui è possibile utilizzare funzioni di event handler, che sono richiamate al verificarsi di determinate condizioni (collisioni, mutazione dello stato e così via...). Le funzionalità di Unity possono essere espanse utilizzando l'Asset Store [11], in cui è possibile scaricare (gratuitamente o a pagamento): plug-in, GameObject e molto altro. Lo sviluppo dell'applicazione è molto semplificato poiché i vari oggetti possono essere semplicemente trascinati nella scena e gestiti mediante l'UI. Quindi

è possibile sviluppare un'applicazione giocattolo anche senza scrivere alcuna riga di codice. Una volta creata l'applicazione, Unity permette di esportarla su diverse piattaforme senza dover effettuare alcuna modifica.

3.5 Integrazione Unity e Vuforia

Avendo scelto come tecnologie target Unity e Vuforia, è necessario integrarle per sfruttare le potenzialità di entrambi. In particolare Unity offre la possibilità di importare l'SDK Vuforia come plug-in, in modo da sviluppare in modo semplice, veloce e intuitivo un'applicazione di realtà aumentata. In figura 3.3, per chiarire come interagiscano l'SDK e l'engine, si riporta il diagramma complessivo delle classi precedentemente in essi mappate. Come sopraccitato, l'utilizzo dell'integrazione di Unity e Vuforia rende molto più semplice lo sviluppo di applicazioni di realtà aumentata. In particolare ciò di cui dovrà preoccuparsi lo sviluppatore sarà esclusivamente della logica di business della propria applicazione. Tutte le funzionalità della realtà aumentata saranno implementate dall'SDK di Vuforia, infatti lo sviluppatore non dovrà preoccuparsi di come addestrare la fotocamera per riconoscere il marker scelto o problemi di tracking quali ad esempio come mostrare gli oggetti virtuali quando vi si gira attorno.

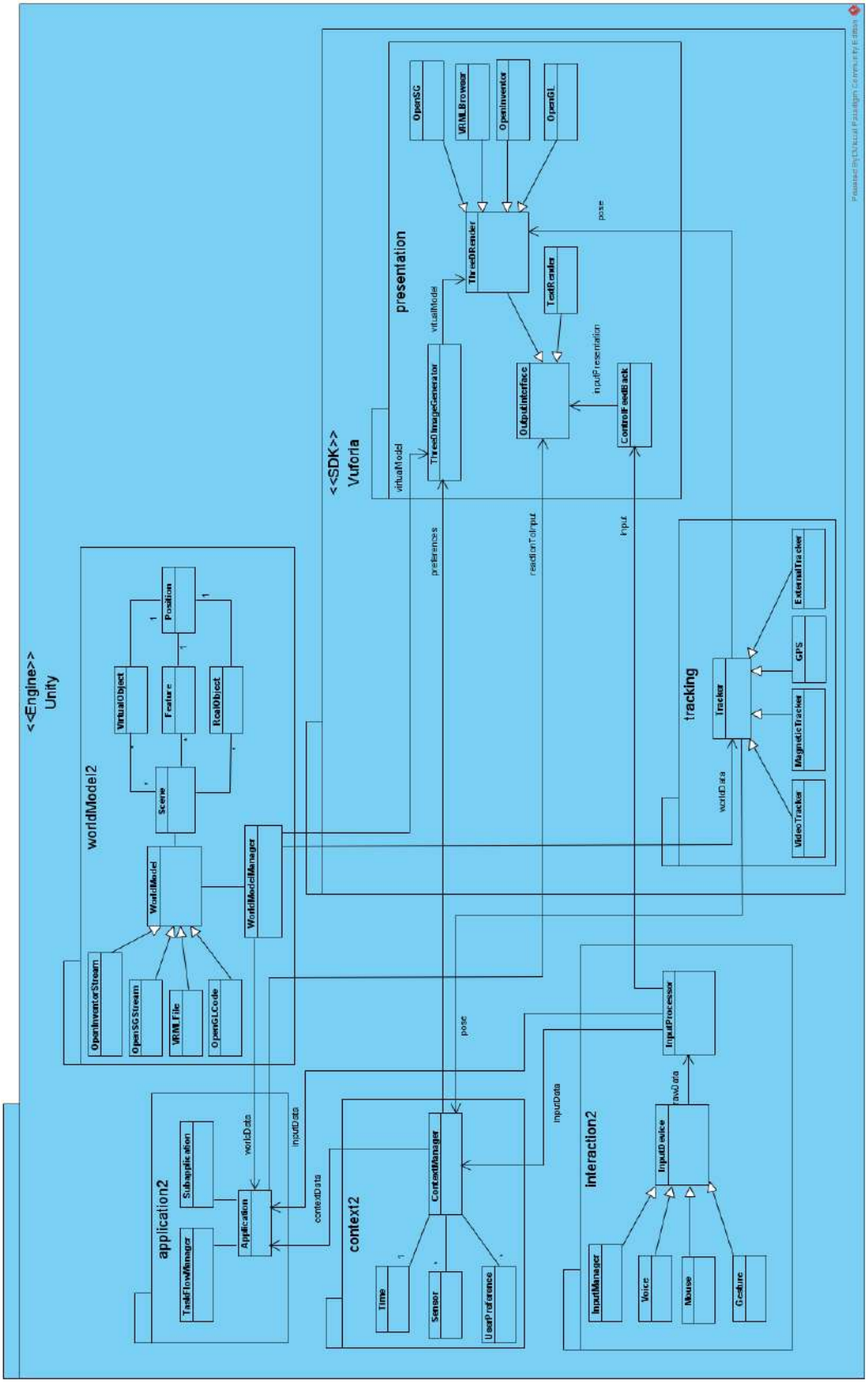


Figura 3.3: Diagramma Unity-Vuforia

Oltre a ciò, Unity consente un semplice assemblaggio e posizionamento degli oggetti virtuali mediante la funzionalità di "drag and drop" e, nel caso in cui si volessero implementare dei comportamenti specifici per ogni oggetto, si può ricorrere alla scrittura di script C#, come descritto nel precedente paragrafo. Per quanto riguarda la versione di Unity, si è optato per la 2018.4.30f1 (rilasciata il 3 Dicembre 2020). Questa versione è stata scelta per uno scopo preciso, ossia perché è la versione più recente che permette di deployare le applicazioni su smartphone (reali o simulati) che incorporano processori x86. Infatti dalla versione 2019 in poi, Unity ha eliminato il supporto per questa tipologia di processori poiché molto poco diffusi. Tuttavia i processori x86 sono utilizzati nei dispositivi simulati perché molto più veloci rispetto a quelli ARM. Come sarà mostrato nei capitoli successivi, le applicazioni, su cui effettuare i tests, saranno deployate su un dispositivo emulato creato mediante AVD Manager. Di seguito saranno mostrati i passi per importare l'SDK di Vuforia in Unity (nel caso si avesse Unity già installato saltare il punto 1).

1 Scaricare Unity dal sito ufficiale [12] ed eseguire l'installazione. Con essa saranno installati Unity Hub, una sorta di tool manger che permette di gestire diverse versioni dell'engine e i relativi plugin, e la versione scelta di Unity;

2 Aprire Unity Hub, scegliere la tab *installs*, cliccare i tre pallini relativi al box della versione Unity, cliccare *Add Modules* dal

menù a tendina, spuntare *Vuforia Augmented Reality Support* e cliccare *Done*. Una volta terminata l'installazione, nel box della versione Unity comparirà una sorta di cubo che rappresenta l'aggiunta del'SDK Vuforia all'IDE. Più precisamente la versione di Vuforia importata sarà la 8.3.8.

Si vuole sottolineare che la procedura d'integrazione varia in base alla versione di Unity utilizzata.

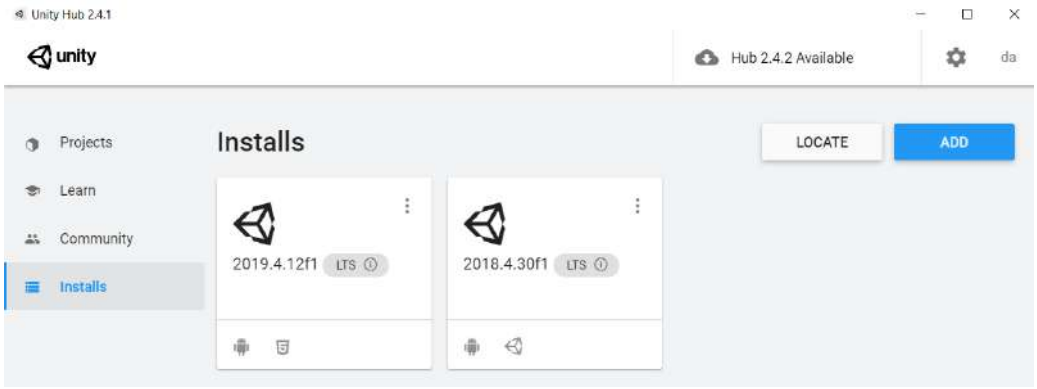


Figura 3.4: Integrazione Unity-Vuforia

Capitolo 4

Testing

In questo capitolo sarà data la definizione di testing con conseguente breve trattazione generale riguardante i diversi livelli di testing. Successivamente ci si concentrerà sul testing di sistema per una precisa tipologia di applicazioni AR, scelte come caso di studio, e si presenteranno i tool utilizzati per effettuarlo.

4.1 Definizione

La definizione di testing ufficiale è fornita da IEEE: *"set of activities conducted to facilitate discovery and/or evaluation of properties of one or more test items. Testing activities could include planning, preparation, execution, reporting, and management activities, insofar as they are directed towards testing."* [44]. Per *test item* s'intende: *"work product that is an object of testing"* [44], per esempio un siste-

ma o elemento software. Quindi il testing è un processo di esecuzione del software al fine di scoprirne i malfunzionamenti. Un malfunzionamento consiste nell'incapacità del software di comportarsi secondo le specifiche ed è osservabile solo mediante esecuzione.

4.2 Tipologie di testing

Il testing può essere effettuato su diverse tipologie di test items. In base a ciò, si differenziano gli obiettivi del test che si sta eseguendo. Di seguito sarà chiarito questo punto mostrando i diversi livelli di test.

4.2.1 Test di unità

Verifica del più piccolo test item, il modulo. Ogni modulo è testato in base alla relativa documentazione di progettazione. Solitamente un modulo necessita di servizi forniti da altri moduli o componenti. Questi sono simulati da Stub o Mock, che forniscono i dati attesi preservando l'isolamento del modulo e di conseguenza del test.

4.2.2 Test d'integrazione

Verifica la correttezza dell'interazione tra più moduli. Un modo per effettuare questo test è costruire in maniera incrementale la struttura del programma, utilizzando i moduli testati mediante test di unità.

Un test di integrazione, di solito, è effettuato per testare un flusso operativo del programma.

4.2.3 Test di Sistema

Ciò che si testa è il sistema completo, ossia il software ultimato e integrato con altri elementi, ad esempio l'hardware. Lo scopo di questo test è verificare che tutti gli elementi del sistema siano correttamente integrati e svolgano correttamente i propri ruoli. Questi test sono condotti basandosi sulle specifiche espresse nella documentazione relativa ai requisiti.

4.2.4 Test di accettazione

Questo test è condotto dall'utente finale e ha l'obiettivo di validare tutti i requisiti. Si divide in:

- α-Test, condotto dal cliente presso lo sviluppatore;
- β-Test, condotto da uno o più utenti nell'ambiente di utilizzo del sistema.

4.3 Tipologia di applicazioni AR

Tra le diverse tipologie di applicazioni AR si è scelta quella più comunemente diffusa, ossia le applicazioni AR per dispositivi mobili. Il

motivo per cui sono più diffuse è abbastanza lampante, poiché oggi la stragrande parte della popolazione possiede uno smartphone su cui è possibile installarle. Di questa categoria di applicazioni ci si concentrerà solo su quelle implementate utilizzando l'IDE Unity e l'SDK Vuforia, come anticipato nel paragrafo 3.5. L'obiettivo che ci si pone è quello di effettuarne il testing di sistema mediante l'utilizzo di appositi tools, in modo da verificarne il corretto funzionamento. Così facendo si eviterà ai tester di eseguire e replicare manualmente i test sulla specifica applicazione, beneficiando di un risparmio dei tempi di lavoro [39] [38].

4.4 Tools per il test di sistema

Il primo passo effettuato è stato quello di cercare in letteratura se vi fosse un tool per il testing di sistema delle applicazioni AR. La ricerca non ha prodotto grossi risultati, ma da essa si è evinto che la stragrande maggioranza delle applicazioni AR sono testate manualmente da tester, che interagiscono con la GUI dell'applicazione da testare. Uno dei framework che supporta il test manuale delle applicazioni è ARCHIE, il quale raccoglie i feedback degli utenti al fine di supportare gli sviluppatori nell'individuazione dei bug e, in base a quanto raccolto, guida i tester nell'effettuazione di test considerati prioritari [54]. Sebbene questo sia un ottimo metodo per effettuare il testing di sistema, rimane comunque centrale la presenza di un tester umano

che effettua manualmente l'interazione con la GUI. Si è quindi pensato di adattare i tools utilizzati per testare le applicazioni mobile alle applicazioni AR mobile, nonostante fra esse siano presenti profonde differenze. Una su tutte è il posizionamento degli oggetti virtuali, dipendente dal comportamento dell'utente e dall'ambiente circostante. La loro elevata variabilità, unita alle condizioni di illuminazioni e alle diverse angolazioni, rende le applicazioni AR molto complicate da testare. La tipologia di test che si effettuerà sarà quella di sistema, poiché non sarebbe possibile replicare, in uno o più unit test, quali combinazioni delle suddette condizioni potrebbero verificarsi. Dalle ricerche condotte, sono emersi due tools che sembrerebbero potersi adattare allo scopo. Entrambi saranno presentati di seguito.

4.4.1 AltUnity Tester

AltUnity Tester è un tool gratuito per il test di sistema di applicazioni realizzate con Unity. E' possibile scaricarlo dall'Unity Asset Store gratuitamente per aggiungerlo al proprio progetto ed eseguire i test all'interno del suddetto IDE. Nel caso in cui si volesse testare l'applicazione al di fuori dell'IDE, è possibile scaricare gratuitamente dal sito ufficiale [1] AltUnity Inspector, che permette di interagire con la gerarchia di oggetti. AltUnity Tester permette di scrivere i test in C#, Python e Java e di eseguirli sia su dispositivi mobili che su PC. Come descritto dalla documentazione ufficiale [13], Alt Unity Tester si

compone di tre moduli:

- AltUnity Server, che permette l'accesso agli oggetti della gerarchia aprendo una socket TCP sul dispositivo che esegue l'applicazione e attendendo la connessione di un AltUnity Client;
- AltUnity Client, utilizzato per connettersi all'AltUnity Server accedendo e interagendo con gli oggetti tramite i tests scritti;
- AltUnity Tester Editor Window, la GUI utilizzata per eseguire i test (solo quelli scritti in linguaggio C#) direttamente dall'IDE di Unity.

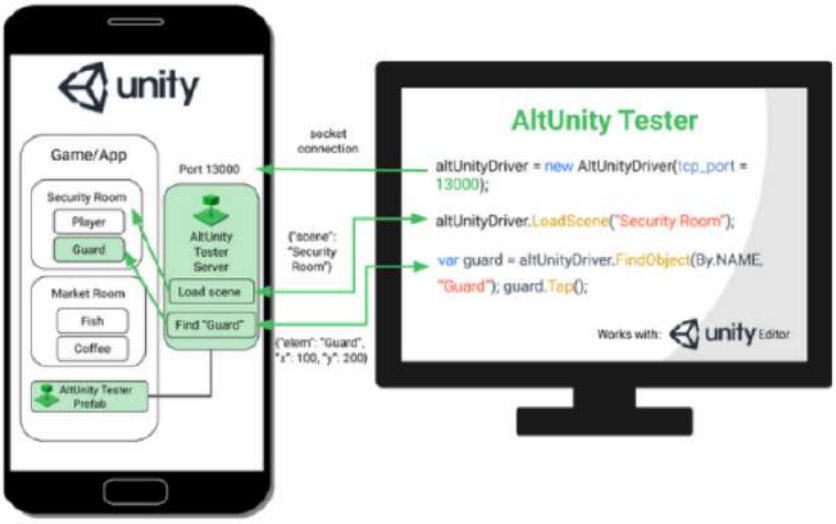


Figura 4.1: AltUnity Tester [1]

4.4.2 AirTestProject

AirTestProject è un framework gratuito per l'automazione di test creato da Netease Games. Come descritto dalla documentazione ufficiale [14], AirTestProject è composto da:

- *AirTest*: tool per il test di sistema di applicazioni, basato sul principio di riconoscimento delle immagini;

- *Poco*: tool che permette di accedere alla gerarchia di oggetti dell'applicazione, per la scrittura di casi di test più avanzati. Per poter accedere alla gerarchia di oggetti è necessaria una previa connessione a Poco-SDK (scaricabile gratuitamente dal repository Github [15]) che deve essere importato all'interno del progetto;

- *AirTestIDE*: ossia lo strumento GUI multipiattaforma con cui è possibile evitare di scrivere i test da riga di comando, semplificandone la loro implementazione mediante una serie di plug-in che permettono di scrivere agevolmente il codice di test;

- *AirLab*: la piattaforma di test in cloud per testare la compatibilità dell'applicazione con diverse tipologie di dispositivi;

AirTestProject si basa sul linguaggio Python e fornisce una serie di API multipiattaforma che includono operazioni quali click, trascinamento, inserimento di testo e così via. Si è scelto di utilizzare il tool Poco piuttosto che AirTest, poiché quest'ultimo utilizza il riconoscimento delle immagini che, in condizioni di luminosità e angolazioni variabili, non risulta essere preciso. Contrariamente Poco accedendo direttamente alla gerarchia degli oggetti riesce ad essere estremamente preciso. Per maggiori informazioni riguardanti le percentuali di rico-

noscimento delle immagini del tool AirTest si rimanda al link della documentazione ufficiale [16]. Per una maggiore semplicità di utilizzo i test saranno scritti in AirTestIDE, che è possibile scaricare gratuitamente dal sito ufficiale [17] e utilizzarlo dopo la decompressione senza alcuna installazione.

4.5 Valutazione tools

A primo acchito potrebbe sembrare che AltUnity Tester e i tools scelti di AirTestProject siano del tutto equivalenti. Per valutarne le potenzialità e l'effettiva adattabilità al testing di applicazioni AR mobile sarà creata una applicazione mobile utilizzando le tecnologie target scelte.

4.5.1 Applicazione

L'applicazione che si è deciso di implementare è concettualmente molto semplice. Si tratta di un'applicazione marker-based che, in caso di riconoscimento del marker, mostra una serie di oggetti virtuali. Il marker scelto è il celeberrimo quadro "La Scuola di Atene" dell'artista Raffaello Sanzio. L'applicazione consiste, dopo l'avvenuto riconoscimento del quadro, nel mostrare un bottone che, se cliccato, presenta titolo e autore a sinistra del quadro. L'implementazione dell'applicazione è stata effettuata utilizzando l'integrazione tra Unity e Vuforia

3.5, sfruttando la funzionalità, messa a disposizione da quest'ultimo, per la creazione di marker personalizzati. Di questa applicazione sono state create due versioni che contengono una piccola ma sostanziale differenza. Entrambe hanno esattamente lo stesso funzionamento, la differenza sta nella tipologia di oggetto Unity usato come bottone. Nella prima versione il bottone è stato realizzato utilizzando un oggetto Unity di tipo "Button", mentre nella seconda versione è stato realizzato utilizzando un parallelepipedo e un testo 3D, entrambi di tipo "GameObject" a cui è stato aggiunto, tramite script, il Raycast. Per Raycast s'intende l'aggiunta all'oggetto della proprietà fisica di ricezione delle collisioni, nella fattispecie i click dell'utente sul bottone. La scelta di utilizzare le due versioni della stessa applicazione è dovuta al fatto che si vuole valutare la capacità, dei tools di testing scelti, di interagire con le diverse tipologie di oggetti, messi a disposizione da Unity. Si fa presente che, per riutilizzare gli stessi script, i nomi degli oggetti, impiegati nelle due versioni dell'applicazione, sono identici. Terminata l'implementazione, entrambe sono state deployate su un dispositivo mobile virtuale, creato con AVD Manager di Android Studio. Alla camera esterna del dispositivo virtuale è stata associata una stanza virtuale, in cui è stato posizionato il quadro che funge da marker. Le specifiche del dispositivo virtuale sono:

- Modello smartphone: Pixel 2;
- Sistema operativo: Android 8.1 Oreo;

- Livello API: 27;
- RAM: 1536 MB;
- Spazio di Archiviazione: 2048 MB interno + 512 MB SD-CARD

Figura 4.2: Due versioni dell'applicazione

4.5.2 Testing con AltUnity Tester

Per Utilizzare AltUnity Tester è necessaria una previa aggiunta al progetto mediante download dall'Asset Store di Unity. Una volta aggiunto al progetto, potrà essere aperta la relativa finestra selezionando "AltUnityTester" dal menù a tendina della tab "Window" dell'IDE. A questo punto dovranno essere aggiunti i file contenenti i test. Questi devo-

no essere necessariamente inseriti in un preciso percorso del progetto, bisognerà creare una cartella "Editor" all'interno di "Asset", successivamente occorrerà cliccare con il tasto destro del mouse su di essa e selezionare "Create" e poi "AltUnityTest". In questo modo sarà creato un file contenente il template per scrivere i test. Dopo aver scritto il test, basterà aprire la finestra "AltUnityTester" nell'IDE, scegliere la piattaforma su cui eseguirlo (il dispositivo Android collegato mediante ADB), selezionare la scena, selezionare uno o più test, lanciare l'applicazione e cliccare su "Run All Tests". Una volta eseguiti tutti i tests, nella console dell'IDE saranno visibili i loro risultati. Quanto detto fino ad ora, riguardo il setting di AltUnity Tester, è disponibile nella documentazione ufficiale [18]. Come anticipato i tests saranno effettuati sulle due versioni dell'applicazione deployate sul dispositivo virtuale. I seguenti scripts di test verificheranno che, al click del bottone, le informazioni relative al quadro saranno visibili/non visibili.

```
using NUnit.Framework;
using System.Threading;

public class testScript
{
    public AltUnityDriver AltUnityDriver;
    public string bottone = "bottone";
    public string testoBottone = "testoBottone";
    public string info = "info";
    //Before any test it connects with the socket
    [OneTimeSetUp]
    public void SetUp()
    {
        AltUnityDriver =new AltUnityDriver();
        AltUnityDriver.LoadScene("SampleScene");
        bottone = "bottone";
        testoBottone = "testoBottone";
        info = "info";
}

    //At the end of the test closes the connection with the socket
    [OneTimeTearDown]
    public void TearDown()
    {
        AltUnityDriver.Stop();
    }

    [Test]
    public void MostraInfoTest()
    {
        AltUnityDriver.WaitForObject(By.NAME, bottone, timeout: 2);
        string testo = AltUnityDriver.FindObject(By.NAME, testoBottone).GetText();
        Assert.AreEqual(testo, "Mostra Info");
        AltUnityDriver.WaitForObjectNotBePresent(By.NAME, info, timeout: 1);
        AltUnityDriver.FindObject(By.NAME, bottone).Tap();
        AltUnityDriver.WaitForObject(By.NAME,info, timeout: 1);
        Assert.IsTrue(AltUnityDriver.FindObject(By.NAME, info).enabled);
    }

    [Test]
    public void NascondiInfoTest()
    {
        AltUnityDriver.WaitForObject(By.NAME, bottone, timeout: 2);
        string testo = AltUnityDriver.FindObject(By.NAME, testoBottone).GetText();
        Assert.AreEqual(testo, "Nascondi Info");
        Assert.IsTrue(AltUnityDriver.FindObject(By.NAME, info).enabled);
        AltUnityDriver.FindObject(By.NAME, bottone).Tap();
        AltUnityDriver.WaitForObjectNotBePresent(By.NAME, info, timeout: 1);
    }

}
```

Figura 4.3: Script di test di AltUnity Tester della prima versione

```
using NUnit.Framework;
using System.Threading;

public class testScript
{
    public AltUnityDriver AltUnityDriver;
    public string bottone;
    public string testoBottone;
    public string info;
    //Before any test it connects with the socket
    [OneTimeSetUp]
    public void SetUp()
    {
        AltUnityDriver =new AltUnityDriver();
        AltUnityDriver.LoadScene("SampleScene");
        bottone = "bottone";
        testoBottone = "testoBottone";
        info = "info";
    }

    //At the end of the test closes the connection with the socket
    [OneTimeTearDown]
    public void TearDown()
    {
        AltUnityDriver.Stop();
    }

    [Test]
    public void MostraInfoTest()
    {
        AltUnityDriver.WaitForObject(By.NAME, bottone, timeout: 2);
        AltUnityDriver.WaitForObject(By.NAME, testoBottone, timeout: 2);
        var testo = AltUnityDriver.FindObject(By.NAME, testoBottone).GetComponentProperty("UnityEngine.TextMesh", "text");
        Assert.AreEqual(testo, "Mostra Info");
        AltUnityDriver.WaitForObjectNotBePresent(By.NAME, info, timeout: 1);
        AltUnityDriver.FindObject(By.NAME, bottone).Tap();
        AltUnityDriver.WaitForObject(By.NAME, info, timeout: 1);
        Assert.IsTrue(AltUnityDriver.FindObject(By.NAME, info).enabled);
    }

    [Test]
    public void NascondiInfoTest()
    {

        AltUnityDriver.WaitForObject(By.NAME, bottone, timeout: 2);
        AltUnityDriver.WaitForObject(By.NAME, testoBottone, timeout: 2);
        var testo = AltUnityDriver.FindObject(By.NAME, testoBottone).GetComponentProperty("UnityEngine.TextMesh", "text");
        Assert.AreEqual(testo, "Nascondi Info");
        Assert.IsTrue(AltUnityDriver.FindObject(By.NAME, info).enabled);
        AltUnityDriver.FindObject(By.NAME, bottone).Tap();
        AltUnityDriver.WaitForObjectNotBePresent(By.NAME, info, timeout: 1);

    }

}
```

Figura 4.4: Script di test di AltUnity Tester della seconda versione

Le differenze tra i due scripts riguardano il modo in cui si ottiene il testo del bottone, ciò è conseguenza della diversa tipologia di oggetti Unity. Eseguendo il test su entrambe le versioni dell'applicazione, si è notato che l'esito sulla prima versione è di successo, mentre nella seconda versione il click sul bottone non è recepito e quindi il test fallisce. Ciò è dovuto al fatto che AltUnity Tester non riesce a interagire (nella fattispecie a cliccare) con i GameObject di Unity. Questo limite è di fondamentale importanza dato che, nelle applicazioni AR, l'inte-

razione con gli oggetti virtuali è basilare.

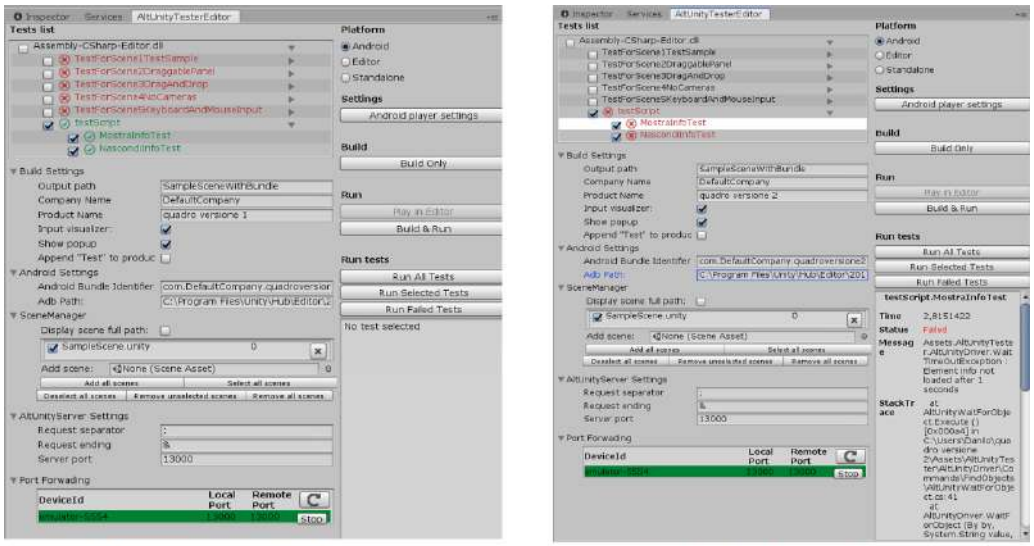


Figura 4.5: Risultati esecuzione tests con AltUnity Tester

4.5.3 Testing con Poco e AirTestIDE

Per effettuare il testing, accedendo alla gerarchia di oggetti Unity, è necessario importare nel progetto la cartella "Unity3D", presente all'interno del Poco-SDK scaricato. A questo punto nel progetto Unity sarà presente la cartella "Unity3D", al suo interno bisognerà eliminare le cartelle "ngui" e "fairygui". Ciò è necessario poiché, in esse, sono presenti gli script per accedere alle GUI di terze parti utilizzate da Unity nelle versioni antecedenti alla 4.6. Attualmente, Unity utilizza esclusivamente il sistema GUI in esso integrato (ugui). Successivamente bisognerà aggiungere lo script "PocoManager" alla ARCamera, che creerà il server che permetterà l'accesso alla gerarchia di oggetti, si-

milmente all'AltUnity Server presentato nel paragrafo 4.4.1. Fatto ciò bisognerà deployare nuovamente l'applicazione sul dispositivo virtuale. Quanto detto fino ad ora, riguardo il setting di Poco, è disponibile nella documentazione ufficiale [19]. Diversamente da AltUnity Tester l'ambiente di scrittura ed esecuzione dei test è esterno a Unity. Come anticipato, l'IDE che sarà utilizzato è AirTestIDE, il cui linguaggio supportato è Python. La prima cosa da fare è collegare il dispositivo all'IDE cliccando il bottone "connect", del dispositivo corrispondente, presente nella finestra "Devices". Ciò permetterà la comunicazione tra AirTestIDE e il dispositivo utilizzando ADB. Fatto ciò si potrà avviare, dall'IDE o dal dispositivo, l'applicazione da testare e, selezionando "Unity" dal menù a tendina della finestra "Poco Assistant", sarà possibile vedere la gerarchia di oggetti dell'applicazione stessa. Quindi, così come AltUnity Tester, AirTestIDE fornisce la possibilità di accedere agli oggetti permettendo la scrittura dei test, nella sostanza, simili a quelli visti nel paragrafo precedente (con le dovute differenze dovute alla diversità di linguaggio).

```python
# -*- encoding=utf8 -*-
__author__ = "Danilo"

from airtest.core.api import *

auto_setup(__file__)


from poco.drivers.unity3d import UnityPoco
poco = UnityPoco();


bottone = poco("bottone");
info = poco("info");
testoBottone = poco("testoBottone").get_text();


assert_equal(bottone.exists(), True, "Bottone non presente sulla scena")
assert_equal(info.exists(),False, "Info compare")
assert_equal(testoBottone, "Mostra Info", "Testo del bottone non atteso");
bottone.click();
assert_equal(info.exists(),True, "Info non compare")


testoBottone = poco("testoBottone").get_text();
assert_equal(bottone.exists(), True, "Bottone non presente sulla scena" )
assert_equal(info.exists(),True, "Info non presente")
assert_equal(testoBottone, "Nascondi Info", "Testo del bottone non atteso");
bottone.click();
assert_equal(info.exists(),False, "Info compare dopo il click di nascondi info")

```

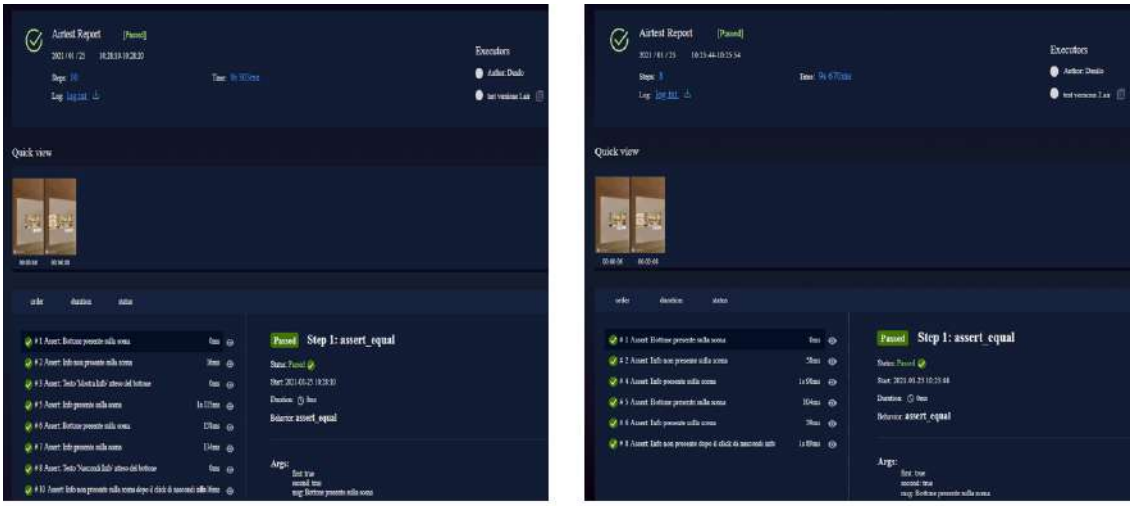
Figura 4.6: Script di test di Poco della prima versione

```python
# -*- encoding=utf8 -*-
__author__ = "Danilo"

from airtest.core.api import *

auto_setup(__file__)


from poco.drivers.unity3d import UnityPoco
poco = UnityPoco();


bottone = poco("bottone");
info = poco("info");


assert_equal(bottone.exists(), True, "Bottone presente sulla scena")
assert_equal(info.exists(),False, "Info non presente sulla scena")
bottone.click();
assert_equal(info.exists(),True, "Info presente sulla scena")


assert_equal(bottone.exists(), True, "Bottone presente sulla scena" )
assert_equal(info.exists(),True, "Info presente sulla scena")
bottone.click();
assert_equal(info.exists(),False, "Info non presente dopo il click di nascondi info")
```

Figura 4.7: Script di test di Poco della seconda versione

Anche qui le differenze tra i due script sono dovute alla diversa tipologia di oggetti Unity. In particolare nel secondo script di test non

si è potuta fare l'asserzione sul testo del bottone. Ciò è dovuto al fatto che il testo è una componente del GameObject "3DText" utilizzato e Poco non fornisce le API per accedervi. Eseguendo il test su entrambe le versioni dell'applicazione, si è notato che sono stati portati a compimento con successo. Il risultato è visibile nel report html, a cui si può accedere selezionando "View Report" nel menù a tendina del tab "Run".



a) Test prima versione dell'applicazione

a) Test seconda versione dell'applicazione

Figura 4.8: Risultati esecuzione tests con Poco

Oltre a questa tipologia di testing, AirTestIDE mette a disposizione anche la funzionalità di "capture & replay", traducendo le azioni in codice con la successiva possibilità di aggiunta di assert. In questo modo si può scrivere il test in modo più veloce ottenendo i risultati mostrati in precedenza.

4.5.4 Confronto

Come si è evinto dei test effettuati, sia AltUnity Tester che i tools di AirTestProject presentano dei limiti. La limitazione che si è ritenuta più grave è quella di AltUnity Tester che non riesce a interagire con i GameObject. Contrariamente, i tools di AirTestProject non presentano alcuna difficoltà di interazione con le varie tipologie di GameObject, ma anch'essi presentano un grave limite che consiste nel non poter accedere ai campi dei componenti costituenti il GameObject. Nonostante ciò si è ritenuto che Poco e AirTestIDE risultino essere abbastanza adattabili al testing delle applicazioni AR mobile e di conseguenza sono stati scelti come tecnologia target con cui provare ad automatizzare i tests.

Capitolo 5

Analisi progetti open-source

In questo capitolo sarà effettuata una ricerca con successiva analisi di progetti AR Mobile open-source, al fine di verificare la presenza di test di sistema e issues. Tuttavia per un'analisi più ampia e dettagliata si rimanda a [55].

5.1 Ricerca progetti

La selezione dei progetti open-source è stata effettuata mediante due ricerche sui repository pubblici di GitHub. Le parole chiave utilizzate per la prima ricerca sono state: Unity, Vuforia, augmented-reality, app; mentre per la seconda: Unity, Vuforia, ar, app. La scelta di queste parole chiave non è casuale, infatti si è voluto limitare la ricerca

solamente ai progetti sviluppati con le tecnologie target scelte nel paragrafo 3.5 e sviluppate per dispositivi mobili. Si fa presente che non si è limitata la ricerca alle versioni di Unity e Vuforia selezionate, questo perché è possibile che si siano effettuati test di sistema su dispositivi mobili fisici. Infatti, se fossero stati condotti test di sistema in questo modo non sarebbe stato necessario limitarsi alle versioni di Unity antecedenti alla 2019. L'ordinamento scelto per i progetti trovati è di tipo "Most stars", ossia ordinamento per numero di stelle decrescenti, dove le stelle rappresentano le preferenze degli utenti.

5.2 Analisi di test e issues

La prima ricerca ha prodotto 101 risultati [20], di questi 34 erano incompleti e quindi solo la restante parte sono stati considerati validi per essere analizzati. La seconda ricerca ha prodotto 146 risultati [21], tuttavia a questi vanno sottratti 36 progetti già presenti nella prima ricerca. Dei restanti 110 progetti, 47 erano incompleti e di conseguenza 63 progetti sono stati ritenuti validi per l'analisi. In definitiva, sono stati analizzati 130 (67+63) progetti e ognuno di essi è stato navigato al fine di scoprire se fossero presenti test di sistema. I progetti trovati sono sostanzialmente giochi di tipo marker-based alcuni per scopo puramente ludico altri per scopo ludico-didattico. Gli sviluppatori di tali progetti sono utenti di GitHub e quindi studenti universitari o persone che divulgano le proprie applicazioni per scopi didattici o ancora per

"sfruttare" come tester coloro che scaricano il progetto, in modo tale da ricevere segnalazioni di eventuali bug. Ciò che si è evinto dall'analisi è che solo 2 di essi presentavano delle classi di test. Tuttavia, tali classi di test, appartenevano alla tipologia unit test, realizzati mediante il tool di testing integrato in Unity, utile per la verifica delle singole funzionalità. Quanto detto è emblematico e conferma ciò che era stato indicato nel capitolo precedente, ossia che i test di sistema sono effettuati manualmente dai tester. Oltre ad indagare la presenza di test, nei progetti ritenuti validi, si è ricercata anche la presenza di issues. Tra le varie issues, si sono tenute in considerazione quelle relative al funzionamento dell'applicazione e si sono tralasciate quelle relative alla documentazione e agli sviluppi futuri. I progetti presentanti delle issues sono risultati essere 2, aventi in totale 7 issues (6, di cui 3 aperte e 3 chiuse, relative al primo e 1, chiusa, relativa al secondo). Le issues sono tutte relative alla GUI, ciò avvalora la tesi che gli utenti che effettuano il download di queste applicazioni fungono da tester. In conclusione il risultato di questa analisi conferma che non vi è una metodologia e né tantomeno dei tools specifici per il testing di sistema di questa tipologia di applicazioni. Quindi la tecnica di adattare un tool di testing di applicazioni mobile ad applicazioni AR mobile, potrebbe essere un'intuizione per dare il la all'inizio del testing automatizzato di questa tipologia di applicazioni.

Capitolo 6

Testing Automation

In questo capitolo sarà definito cosa s'intende per testing automation, sarà proposta una metodologia per modellare le applicazioni AR e come scaturire da essa la test suite. Sarà inoltre mostrata la metrica utilizzata per la valutazione della test suite, con il relativo strumento realizzato per calcolarla, e in fine sarà descritta l'analisi mutazionale al fine di utilizzarla per constatare la qualità della test suite.

6.1 Definizione

Avendo introdotto il test nel capitolo 4, si effettua un ulteriore passo avanti definendo cosa s'intende per testing automation. Volendo tradurre in italiano tali parole si otterrebbe automazione del test. Ma cosa vuol dire automazione? Per automazione s'intende un processo mediante cui possiamo, anche parzialmente, rendere automatico un

processo manuale. Le aree di intervento su cui maggiormente si basa il testing automation sono:

- Automazione nella generazione dei casi di test, ha l'obiettivo di ridurre i costi e i tempi relativi alla fase di test design. Esistono diverse metodologie per ottenere tale automazione, una di queste è il Model Base Testing che consente la generazione di una test suite a partire da un modello. Tale metodologia sarà trattata nella sezione 6.2.1;

- Automazione nell'esecuzione dei casi di test, ha l'obiettivo di evitare l'interazione del tester con l'applicazione da testare. In altre parole l'esecuzione manuale di questa tipologia di test comporta un notevole impiego di tempo e risulta essere ripetitiva, quindi, dopo un certo periodo, il tester umano potrebbe perdere lucidità ed effettuare il test in modo più approssimativo. Quest'area del testing automation mira a risolvere la suddetta problematica poiché, una volta scritto lo script di test, sarà eseguito un qualsiasi numero di volte senza alcun intervento umano e mantenendo sempre la stessa precisione. Tale automazione sarà implementata utilizzando i tools, nella fattispecie Poco e AirTestIDE, di cui sono state valutate le potenzialità nel paragrafo 4.4.2;

- Automazione nella valutazione dell'esito dei casi di test, ha lo scopo di valutare automaticamente la riuscita di un caso di test.

Per fare ciò è necessario che il test case sia stato preventivamente definito e sia fruibile un metodo per la sua valutazione (ad esempio gli assert). Nel caso in esame, la sinergia di Poco e AirTestIDE fornisce al tester una serie di assert in grado di valutare l'esito del test case eseguito.

6.2 Metodologia di testing

Ciò che si farà sarà applicare una metodologia di testing alle applicazioni di realtà aumentata, in modo da definire chiaramente quali test dovranno essere effettuati. Per raggiungere questo obiettivo si ricorrerà al Model Based Testing (MBT) utilizzando come modello la macchina a stati finiti, definendo in questo modo una sottocategoria del MBT, ossia lo State Based Testing (SBT). Di seguito MBT e SBT saranno descritti in generale per poi precisare come applicare tale metodologia di testing ai casi di studio.

6.2.1 Model Based Testing

Come si può intuire dalla nomenclatura, il Model Based Testing è una tecnica di test software in cui il comportamento del sistema, che si vuole testare, è confrontato con quello previsto e teorizzato precedentemente da un modello. Per modello s'intende una descrizione formale del sistema con la finalità di comprenderne e specificarne i compor-

tamenti sulla base degli input ricevuti. Da tale formalizzazione sarà possibile scaturire i tests avendo ben chiari gli input e comportamenti del sistema da testare (SUT, System Under Test). Tale tipologia di testing rende anche possibile l'automazione della codifica dei tests, scaturita a partire dall'elaborazione dei modelli creati dagli sviluppatori o dai tester. Il MBT porta con sé un serie di benefici che ne hanno incrementato il successo negli ultimi decenni. In particolare, avendo creato un modello del sistema, la manutenzione della test suite sarà molto più semplice (nel caso in cui si utilizzino dei tools per la codifica automatica, al cambiamento del modello corrisponderà la modifica dei tests ad esso associati) con conseguente risparmio di tempo e miglioramento della copertura dei test. Tuttavia, per attuare tale metodologia, sono previsti dei costi iniziali dovuti alla formazione del personale al fine di diventare esperti nel MBT [46]. Quest'ultimo abbraccia una famiglia molto ampia di approcci, che può declinarsi in modi diversi in base ai contesti. Infatti, i modelli messi a disposizione nel MBT spaziano da quelli grafici a quelli testuali, anche se quelli maggiormente utilizzati sono i primi [46] e per avere una panoramica completa delle varie tipologie di questa categoria si rimanda a [22]. La scelta del modello da utilizzare non è univoca ed è atta a descrivere nel miglior modo possibile le funzionalità del SUT. Tuttavia, per modellare il sistema con le sue funzionalità, è necessario un previo reperimento delle informazioni che lo descrivono. Tali informazioni sono contenute nel documento del-

le specifiche dei requisiti (SRS, Software Requirements Specification), o, in mancanza di esso, in un documento in linguaggio informale che abbia gli stessi scopi. In quest'ultimo caso potrebbero insorgere una serie di problematiche, relative alla definizione delle funzionalità del sistema, dovute alle ambiguità del linguaggio utilizzato.

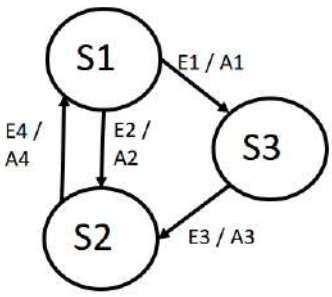
6.2.2 State Based Testing

Lo State Based Testing è un particolare MBT che utilizza come modello la macchina a stati finiti (FSM, Finite State Machine). Quindi per ben comprendere lo SBT è necessario approfondire il modello su cui esso si basa. Come teorizzato da Binder in [30], la FSM rappresenta un sistema il cui output è determinato sia dallo stato corrente in cui si trova che dall'input passato. Di conseguenza, in base allo stato in cui si trova il sistema, si definiscono gli input che è possibile fornirgli e per questa ragione è possibile che, in stati diversi, uno stesso input non sia accettato oppure sia accettato ma produca output differenti. Per formalizzare tali concetti si definiscono i blocchi fondamentali con cui rappresentare la FSM:

- state, rappresenta un possibile stato del sistema e ne descrive il comportamento in caso di input;
- transition, passaggio da uno stato all'altro consentito da un evento;

- event, input fornito;
- action, output che segue a un evento.

E' importante sottolineare che il modo in cui gli event sono generati non fa parte del modello ed, nel momento in cui se ne verifica uno, è ammessa una sola transition con il conseguente spostamento da uno stato di partenza a uno stato di arrivo. I componenti sopraccitati sono utilizzati per la rappresentazione grafica o tabellare della FSM. Nel primo caso: gli state sono rappresentati da riquadri, le transition da frecce orientate mentre gli event e le action da una didascalia associata all'arco; nel secondo caso: sul lato sinistro della tabella sono elencati tutti gli stati, in alto sono presenti gli eventi mentre le celle rappresentano lo state di arrivo dopo che si è verificato l'event. Di seguito si mostra un esempio per entrambe le rappresentazioni della FSM.



a) Rappresentazione grafica

	E1	E2	E3	E4
S1	S3	S2	-	-
S2	-	-	-	S1
S3	-	-	S2	-

a) Rappresentazione tabellare

Figura 6.1: Rappresentazioni FSM

Ciò che ora è necessario capire è come modellare correttamente una FSM. Per questa ragione, di seguito saranno definite una serie di

proprietà che permetteranno l'ottenimento di un modello corretto e consistente.

Stato iniziale e finale

In ogni FSM è necessaria la presenza di uno stato iniziale e uno finale. La caratteristica dello stato iniziale è che deve avere transizioni esclusivamente in uscita, viceversa lo stato finale deve avere transizioni solo in ingresso. Entrambi questi stati sono solitamente rappresentati con dei "pseudo-stati" assimilabili a cerchi di colore nero, come mostrato nella figura di seguito.

Figura 6.2: Stato iniziale e finale

FSM minima

Questa proprietà permette di assicurarsi che la FSM costruita sia minima, cioè non presenti stati equivalenti. La definizione di stati equivalenti è la seguente: *"Due o più stati sono equivalenti se tutte le possibili sequenze di event ad essi applicati producono lo stesso comportamento"* [30]. La presenza di stati equivalenti comporta un modello ridondante e probabilmente incompleto e scorretto.

Reachability

la proprietà della reachability (raggiungibilità) afferma che: *"Dato uno stato S_f e uno stato S_t, si dice che lo stato S_f è raggiungibile dallo stato S_t se esiste una sequenza di event che permette tale spostamento"* [30]. Se ogni stato, ad esclusione di quello iniziale e di quello finale, ha almeno una transizione in ingresso e in uscita (che non sia un cappio, ossia una transizione avente stesso nodo sorgente e destinazione) è verificata tale condizione e ciò significa che dallo stato iniziale è possibile raggiungere uno qualsiasi degli altri stati. In questo modo si evita la comparsa di:

- Dead state, ossia uno stato che non presenta event di uscita quindi, una volta giunti in esso, non è possibile lasciarlo;
- Dead loop, un loop di stati da cui non è possibile uscire e in cui si rimane "intrappolati";
- Magic state, ossia uno stato in cui non vi è alcuna transizione d'ingresso ma solo di uscita (fatta eccezione per lo stato iniziale).

Definizione transition

Per permettere lo spostamento da uno stato all'altro è necessario definire le transition. Per fare ciò è necessario una previa enumerazione di tutti gli event e le action del sistema, così facendo sarà possibile associare, almeno una volta, ognuno di essi a una transition. Con tale

procedimento ci si assicurerà che tutti gli spostamenti da uno stato all'altro, con le relative conseguenze, siano stati modellati

Condizioni di guardia

Per le transition è possibile specificare delle condizione di guardia. Tali condizioni sono utili quando, all'interno del medesimo stato, uno stesso event può comportare una transizione verso due stati (differenti o il medesimo stato di partenza e uno diverso). In questa particolare situazione, è possibile associare agli event le condizioni di guardia che attivano esclusivamente una delle due transizioni. Infatti la guardia non è altro che un predicato, associato a un evento, che deve essere soddisfatto. In altre parole la transizione da effettuare è scelta in base all'event e alla soddisfazione della condizione di guardia ad esso associato.

Invariante di stato e post-condizioni

Riguardo la scelta degli stati, è necessario capire quanti sono indispensabili nel modello per evitare che il loro numero sia eccessivamente elevato. Per evitare ciò si ricorre all'utilizzo dell'invariante di stato correlato alla valutazione delle post-condizioni. Per invariante di stato s'intende: "un'asserzione che definisce uno stato valido" [30], mentre per post-condizioni ciò che ci si aspetti sia verificato dopo un event. In altre parole l'invariante è la condizione generale che verifi-

ca lo stato in cui ci si trova, mentre le post-condizioni definiscono la particolare condizione in cui ci si trova (action) dopo che si è verificato un event. Correlando questi due concetti è possibile semplificare il numero di stati. Per chiarire meglio questo concetto di seguito si propone un esempio. Si supponga di dover modellare un'applicazione che presenti due bottoni B1 e B2. A ognuno di essi è associato un widget che compare in modo esclusivo sullo schermo nel momento in cui il relativo bottone è cliccato. Nel momento in cui si avvia l'applicazione il widget mostrato di default è quello relativo al bottone B1. La FSM rappresentante questa applicazione si potrebbe immaginare essere quella mostrata in figura 6.3.

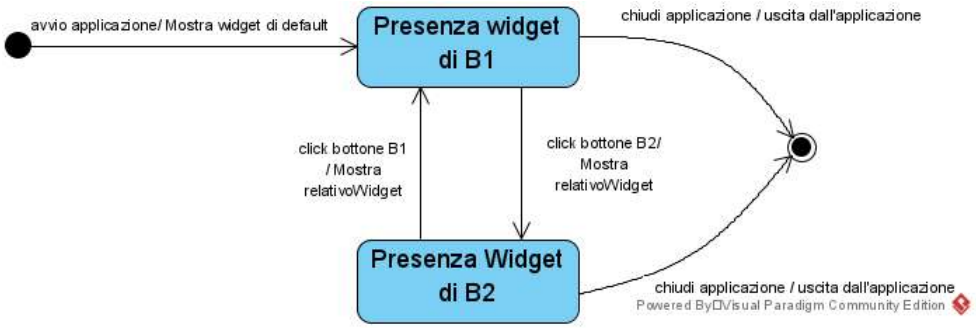


Figura 6.3: FSM di esempio sbagliata

Tuttavia questa rappresentazione risulta essere errata poiché ha un numero di stati maggiore rispetto al necessario. Infatti applicando quanto detto precedentemente riguardo l'invariante di stato e le post-condizioni si ha rispettivamente che:

- la condizione generale che verifica lo stato è la comparsa del widget sulla scena;

- le seguenti due situazioni particolari: se cliccato B1 comparsa del relativo widget; se cliccato B2 comparsa del relativo widget.

Quindi ciò cambia nelle due post condizioni è solo la tipologia di widget che compare e di conseguenza non risulta utile avere uno stato per ognuno di essi, ma piuttosto un solo stato di presenza widget che verifica la correttezza di ciò che compare in base al bottone cliccato. Ciò è confermato dal fatto che l'invariante di stato presenta una sola condizione, ossia la verifica della comparsa del widget sulla scena. In figura 6.4 si mostra la versione corretta della FSM.

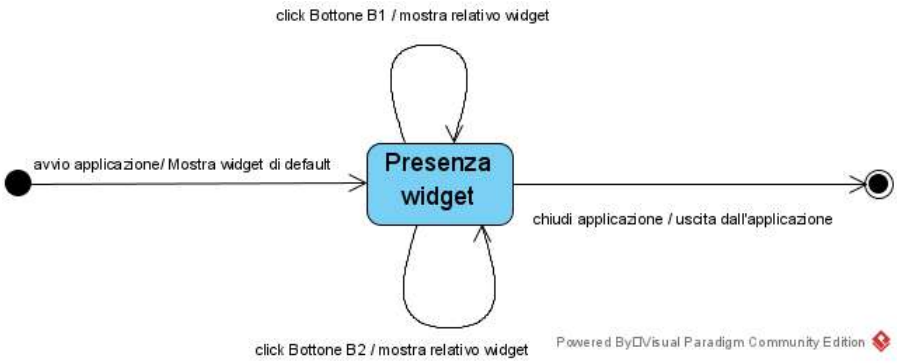


Figura 6.4: FSM di esempio corretta

6.2.3 Criteri di copertura per FSM

Una volta costruita la FSM è necessario far scaturire da essa i test da attuare sul SUT. Per questo motivo si definiscono i criteri di copertura, che si prefiggono lo scopo di selezionare un insieme di test tale da garantire che siano testate le funzionalità del SUT. In particolare, di seguito saranno illustrati una serie di criteri di copertura tra quelli

maggiormente utilizzati [51]. Prima di mostrare i vari criteri è necessario precisare cosa s'intende con il termine "path", utilizzato nelle loro seguenti descrizioni. Tale termine indica un percorso da uno stato inziale a uno finale e comprende i vari stati intermedi attraversati e le annesse transizioni effettuate.

All-States Coverage

Questa tipologia di copertura implica che tutti gli stati, presenti nel modello, risultino coperti almeno una volta. Tale criterio rappresenta il requisito minimo di qualità anche se è considerato il criterio più debole fra tutti quelli che saranno presentati. Ciò è dovuto al fatto che si è focalizzati nel verificare la correttezza degli stati attraversati mediante uno dei possibili paths.

All-Transitions Coverage

Tale criterio di copertura verifica che tutte le transizioni siano effettuate almeno una volta. Questo è quello maggiormente utilizzato poiché fornisce una garanzia di attivazione di ogni evento che permette la transizione di stato. Si può intuire che, attivando tutte le transizioni almeno una volta, saranno visitati tutti gli stati.

All-One-Loop-Paths Coverage

Tale copertura copre tutti i paths che contengono al massimo un loop. Per path contente loop s'intende un percorso che contiene al più uno stato ripetuto due volte. Simile a questo criterio è il All-Loop-Free-Paths Coverage che copre i paths contenti loop attraversati almeno una volta. Tuttavia tale criterio spesso non copre né tutti gli stati né tutte le transizioni e inoltre effettuando una All-One-Loop-Paths Coverage saranno esaminati tutti i paths della All-Loop-Free-Paths Coverage.

All-Paths Coverage

Questa tipologia di copertura esige che ogni path eseguibile sia attraversato almeno una volta. Si può facilmente intuire che effettuare una copertura di questo tipo significherebbe testare il sistema in tutti i casi possibili. Nella maggioranza dei modelli tutti i possibili paths risultano essere infiniti e per tale ragione risulterebbe impossibile percorrerli tutti.

6.2.4 Modellazione

La motivazione per cui nella sezione precedente è stato approfondito un particolare modello del MBT, ossia l'SBT, è dovuto al fatto che ben si presta alla modellazione delle GUI delle applicazioni di realtà aumentata e, più nello specifico, alle applicazioni AR marker-based (come sarà mostrato nel successivo capitolo). In particolare, ciò che

ha portato a questa scelta è da identificare nelle interazioni per la modifica della scena AR mostrata. Per interazioni s'intende:

- Il riconoscimento/disconoscimento del marker con la conseguente comparsa/scomparsa degli oggetti virtuali;
- L'interazione con un oggetto virtuale per la modifica della scena.

Come si può intuire, il riconoscimento/disconoscimento del marker o l'interazione con uno degli oggetti virtuali rappresentano gli eventi trigger (event) che scatenano la modifica della scena attuale. Alla luce di quanto detto la FSM generale di un'applicazione AR marker-based avrà:

- uno stato iniziale, definito "stato di attesa", che attenderà la comparsa di uno o più marker per la creazione della scena;
- un numero di transition pari almeno al doppio dei marker poiché a ognuno di essi corrisponderà l'event di riconoscimento e disconoscimento.

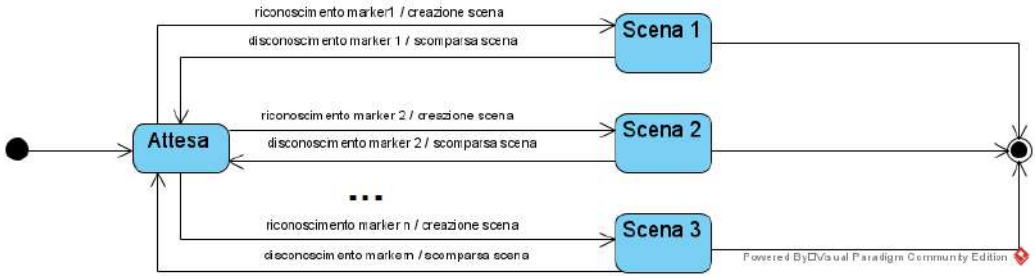


Figura 6.5: FSM generale

In figura 6.5 è mostrata la FSM generale descritta in precedenza. Si noti che nell'immagine non sono stati inseriti i possibili altri stati raggiungibili da eventuali trigger di ognuna delle scene.

6.2.5 Definizione della test suite

Dopo aver modellato il sistema e di conseguenza aver ottenuto la FSM, è possibile applicare su di essa i criteri di copertura mostrati nella sezione 6.2.3 al fine di scaturire la test suite. Come si può immaginare, gli obiettivi e il numero di test da effettuare varieranno in base al criterio di copertura scelto. Tra i criteri proposti quello che solitamente non è applicato è All-Paths Coverage poiché, se la FSM presentasse cicli e/o cappi, il numero di path possibili risulterebbe essere infinito. Per quanto riguarda gli altri criteri, saranno applicati tutti in modo da constatare quali fra essi risulterà essere il migliore per coprire la FSM dell'applicazione che si sceglierà di testare. La test suite, relativa al criterio selezionato, sarà presentata sotto forma di tabella contenente i test case da effettuare. Ogni tabella presenterà i seguenti campi:

- Id: identificativo dello specifico caso di test;
- Descrizione: breve descrizione, in linguaggio naturale, del test che dovrà essere effettuato;
- Pre-condizione: indica le condizioni preliminari che devono essere soddisfatte prima di eseguire il test;

- Post-condizione: indica le condizioni che devono essere verificate dopo l'esecuzione del test;
- Stati: indica quali stati sono stati coperti nel test case;
- Transizioni: indica quali transizioni sono state coperte nel test case.

A tali campi dovrà esserne aggiunto un altro, ossia "Esito". La colonna relativa a questo campo potrà essere compilata solo dopo l'effettuazione dei test case.

6.3 Valutazione test suite

Il criterio scelto per valutare l'efficacia delle test suite selezionate è la copertura del codice. Sfortunatamente AirTestIDE non mette a disposizione nessuno strumento in grado di fare valutazioni riguardo tale copertura. Di seguito si mostra come si è ovviato a tale problema presentando il metodo utilizzato.

6.3.1 Metodo delle sonde

Come anticipato AirTestIDE non mette a disposizione una funzionalità di copertura del codice. Per questa ragione si è ricorso al metodo delle sonde, che consiste nell'aggiungere delle istruzioni (sonde) agli script, al fine di generare log e scriverli su file di testo (.txt). Tali istruzioni

sono posizionate nei vari rami degli script, cosicché, una volta attivati durante l'esecuzione dell'applicazione, si ha coscienza di quali rami del codice sono stati eseguiti facendosi un'idea della copertura del codice ottenuta. Per la realizzazione di tale metodo è necessario aggiungere al progetto da testare lo script "SondaManager", mostrato in figura 6.6.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;

public class SondaManager : MonoBehaviour
{
    private static bool primoAccesso = true;
    private static string path;

    public static void inserisciSonda(string sonda)
    {
        if (primoAccesso)
        {
            path = Application.persistentDataPath + "/LogFile.txt";
            primoAccesso = false;
        }

        File.AppendAllText(path, sonda + "\n");

    }
}
```

Figura 6.6: Script SondaManager

In esso è presente un unico metodo, "inserisciSonda" a cui è passato come parametro la stringa contenente il testo della sonda. Il funzionamento del metodo è molto semplice e consiste: se si tratta del primo accesso nel definire il path del file di log e la scrittura su di esso della sonda passata come parametro; nel caso di successivi accessi consiste esclusivamente nell'inserire in coda al file la sonda ricevuta come pa-

rametro. Si fa presente che il path su cui sarà salvato il log è relativo al dispositivo mobile su cui è eseguita l'applicazione. In particolare il suddetto log sarà salvato nella memoria interna del dispositivo all'interno della cartella "files", contenuta nella directory relativa al progetto a sua volta contenuta nella cartella "data" di Android. Definita la classe "SondaManager", il suo metodo è stato richiamato negli script già presenti nel progetto. Come anticipato per ogni script è stata inserita una sonda per ognuno dei possibili flussi, rappresentati dai costrutti selettivi e iterativi di ognuno dei metodi. Per distinguere le attivazioni dei diversi flussi del codice si è pensato di definire le sonde secondo il seguente template:

NomeScript.NomeFunzione.costruttoECondizione.CostruttoInnestato

Il principio è quello di creare una sorta di gerarchia utilizzando i punti per innestare funzioni e costrutti. Per chiarire il concetto di seguito sono riportati due esempi:

```
void Start()
{

    // Assign the above variable "rb" to the GameObject's RigidBody component
    rb = GetComponent<Rigidbody>();

    // Assign the above variable "anim" to the GameObject's Animation component
    anim = GetComponent<Animation>();
    SondaManager.inserisciSonda("AnimalsController.start");

}
```

Figura 6.7: Esempio sonda n.1

```
void Update()
{
    SondaManager.inserisciSonda("AnimalsController.update");
    // Get the horizontal axis from the joystick
    // (CrossPlatformInput is the library which controls the joystick)
    float x = CrossPlatformInputManager.GetAxis("Horizontal");

    // Get the vertical axis from the joystick
    float y = CrossPlatformInputManager.GetAxis("Vertical");

    // Create a new vector, which will hold the position of the joystick
    Vector3 movement = new Vector3(x, 0, y);

    // Rate of speed the animals will be moving
    rb.velocity = movement * 0.15f;

    // If joystick's x and y values are different from 0
    // i.e. joystick has been moved
    if (x != 0 && y != 0)
    {
        // Move the animal in the direction of the joystick
        transform.eulerAngles = new Vector3(transform.eulerAngles.x, Mathf.Atan2(x, y) * Mathf.Rad2Deg, transform.eulerAngles.z);
        SondaManager.inserisciSonda("AnimalsController.update.if (x != 0 && y != 0)");
    }
```

Figura 6.8: Esempio sonda n.2

Nel primo esempio di figura 6.7, la sonda è composta solo dal nome dello script e dal nome della funzione poiché in essa non sono presenti costrutti. Nel secondo esempio di figura 6.8, invece è presente un costrutto "if" all'interno della funzione, di conseguenza la sonda sarà composta dal nome dello script, nome funzione e costrutto.

6.3.2 Analisi Log

Predisposta la creazione del log, diventa necessario elaborarlo al fine di estrarre le informazioni riguardanti la copertura dei rami del codice. Si è quindi pensato di realizzare un programma Java, denominato "Analisi Log", in grado di effettuare tale elaborazione. Il suddetto programma è in grado di eseguire le seguenti funzionalità:

- Importare automaticamente dal dispositivo e caricare il file di log creato oppure esclusivamente di caricarlo da PC se non è necessario l'import da dispositivo.

- Caricare gli script su cui effettuare l'analisi;
- Analizzare log e script.

Per facilitare ulteriormente l'utilizzo di "Analisi Log" si è scelto di realizzare un'interfaccia grafica. Di seguito saranno dettagliate le funzionalità sopraccitate e le relative GUI.

Import e caricamento file di Log

All'apertura di "Analisi Log" la GUI che compare è mostrata in figura 6.9.

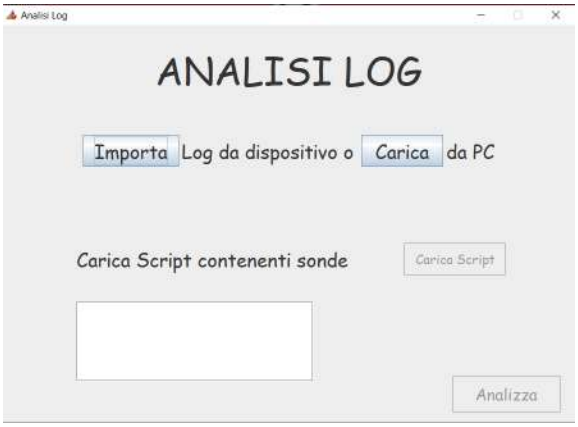


Figura 6.9: GUI iniziale Analisi Log

Le uniche interazioni possibili sono i click sul bottone "Importa" o "Carica". Come si può intuire entrambi caricheranno il file di Log in memoria, tuttavia la differenza sta nel modo in cui realizzano tale funzionalità:

- "Importa", ha il compito di importare il file dal dispositivo mobile e successivamente di caricarlo in memoria. Per quanto riguarda

l'import, è necessario che l'utente specifichi il nome del package dell'applicazione e il nome da assegnare al file di log da importare. Il nome del file può contenere esclusivamente lettere e numeri senza spazi. All'avvenuto inserimento del nome del package e del nome del log che soddisfi i precedenti requisiti, il bottone di "Import" diventerà cliccabile. Prima di cliccare il bottone è necessario assicurarsi che il dispositivo (reale o emulato) collegato al PC sia in esecuzione. Cliccando il bottone, si effettua l'import del log mediante l'esecuzione di comandi shell ADB. Tali comandi permettono di verificare la presenza del package inserito, rinominare il file, importarlo in Desktop ed eliminarlo dal dispositivo. Dopo queste operazioni il log viene caricato in memoria, salvando l'intero contenuto del file. La GUI relativa a quanto descritto è mostrata in figura 6.10;

- "Carica", ha il solo compito di caricare in memoria il file di log, già presente su PC. Al click di tale bottone è possibile navigare le directory per scegliere il file di log da caricare. I file mostrati sono solo di tipo ".txt" coerentemente rispetto al formato del file di log. Una volta scelto il file, sarà caricato in memoria salvandone l'intero contenuto.

Una volta avvenuto l'eventuale import e il caricamento del log comparirà un messaggio di conferma e sarà possibile importare gli script, grazie allo sblocco del bottone "Carica Script".

Figura 6.10: GUI import Analisi Log

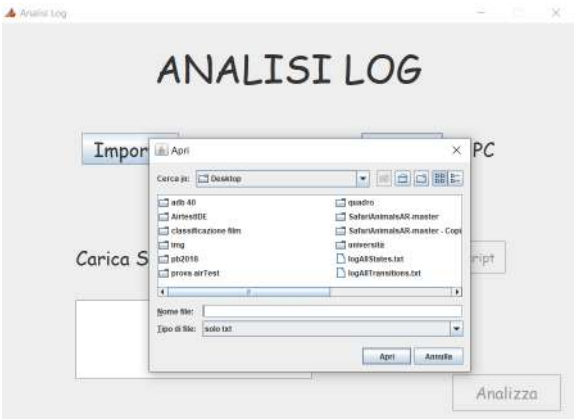


Figura 6.11: GUI carica Analisi Log

Caricamento Script

Una volta importato il file di log sarà possibile caricare gli script che saranno successivamente analizzati. L'operazione di caricamento è effettuata mediante il click sul bottone "Carica Script", il quale permette di navigare le directory per scegliere lo script da caricare. E' possibile ripetere questa operazione più volte. Tutti gli script caricati saranno mostrati nell'area in basso a sinistra, dove comparirà il nome dello script caricato e un bottone di eliminazione, nel caso in cui se ne vo-

lesse eliminare uno. Il caricamento in memoria dello script consiste nel salvare solo le sonde presenti al suo interno, in modo tale da aver chiaro quali rami sono presenti all'interno dello specifico script. All'avvenuto caricamento di almeno uno script sarà possibile cliccare il bottone "Analizza" per l'effettuazione dell'analisi sul log e lo/gli script.

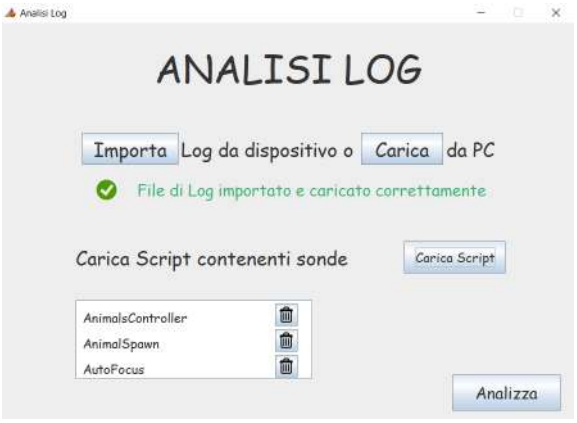


Figura 6.12: GUI carica script Analisi Log

Analisi log e script

Caricati log e script è adesso possibile passare alla loro analisi. La GUI che presenta l'analisi dei file caricati è mostrata in figura 6.13. In essa sono presenti due aree di testo:

- La prima in alto presenta il numero di esecuzioni per ogni ramo contenuto nel log. In altre parole indica quante volte è stato eseguito quel particolare ramo, durante tutto il tempo in cui è stata in esecuzione l'applicazione. Tale risultati si sono ottenuti contando le occorrenze di ogni ramo distinto presente all'interno del file di log caricato;

- La seconda mostra la percentuale di copertura dei rami del codice relativa ad ognuno degli script caricati. Tale percentuale è ottenuta contando quanti e quali rami sono presenti all'interno del file di log, utilizzando la seguente formula:

$$c = \frac{rc}{rt} * 100$$

Dove con rc si indica il numero di rami contenuti nel file di log e quindi coperti, mentre con rt si indica il numero di rami totali presenti nello script. Per evidenziare maggiormente la percentuale di copertura, sono state impostate delle soglie in base alle quali varia il colore del nome e della percentuale mostrati. In particolare per una copertura minore o uguale del 30% il colore associato è il rosso, superiore al 30% e inferiore al 75% il colore associato è l'arancione e, infine, pari o superiore al 75% il colore associato è il verde. Per ogni script che non raggiunge la copertura totale (100%) compare un bottone che, se cliccato, offre la possibilità di visualizzare quali rami di codice non sono stati eseguiti.

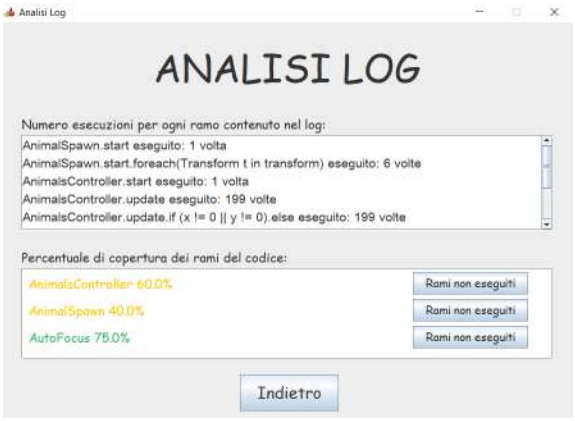


Figura 6.13: GUI analisi Analisi Log

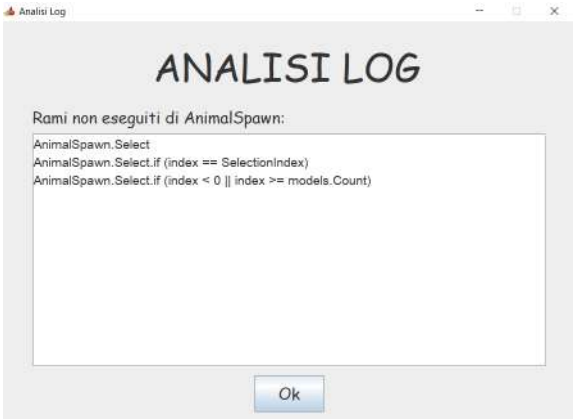


Figura 6.14: GUI rami non eseguiti Analisi Log

6.3.3 Analisi mutazionale

Al fine di confrontare la qualità delle test suite ottenute si introduce l'analisi mutazionale. L'analisi mutazionale consiste nell'iniettare difetti all'interno del SUT supposto corretto, così facendo sarà possibile confrontare la capacità di rilevazione dei difetti delle diverse tecniche di generazione di test suite. Per prima cosa è fondamentale capire quali possano essere errori verosimili, in modo tale da definire un fault model. Tale modellazione può essere fatta basandosi: o sull'esperienza

di colui che propone il testing mutazionale o su un'analisi statistica dei difetti rilevati in altri software. Ottenuto tale modello, è necessario associare a ognuno degli errori un operatore di mutazione che ha l'obiettivo di introdurre un difetto all'interno del SUT, realizzando così un'operazione di fault injection. Gli operatori utilizzati sono molto generici e indipendenti rispetto al linguaggio e al dominio dell'applicazione, ad esempio: sostituzione di un'operazione aritmetica con un'altra, sostituzione di un valore booleano, sostituzione di un operatore relazionale e via dicendo. Conseguentemente a un'operazione di fault injection si avrà una nuova versione del SUT denominata mutante. Come si può intuire, ogni fault injection al SUT, supposto corretto, determinerà un nuovo mutante. Una volta ottenuti i mutanti, su ognuno di essi sarà eseguita la test suite al fine di valutarne se essa sarà in grado di scoprire il difetto introdotto. Per fare ciò è necessaria la definizione dell'oracolo, che risulta essere il metro di paragone definendo i risultati attesi (in questo caso il comportamento del sistema prima della fault injection). Ciò significa che se l'esito del test condotto sul mutante non coincide con l'oracolo allora la mutazione è stata scoperta e il mutante è stato ucciso (killed). Altrimenti, se quanto detto in precedenza non avviene, la mutazione non è stata scoperta, di conseguenza il mutante si ritiene sopravvissuto e il SUT è ritenuto erroneamente privo di malfunzionamenti. Dall'esecuzione della stessa test suite sui diversi mutanti, è possibile calcolarne l'efficacia, definita

TER (Test Effectiveness Ratio), utilizzando la seguente formula:

$$TER = \frac{km}{tm}$$

Dove per km s'intende il numero di mutanti scoperti (killed mutants), mentre per tm il numero totale di mutanti (total mutants). Come si può intuire, il valore ottenuto dalla suddetta formula varia tra 0 e 1, in particolare più è vicino a 1 maggiore sarà la qualità della test suite poiché si avrà maggiore fiducia nella sua capacità di scoprire difetti (supponendo che i difetti iniettati siano rappresentativi di quelli reali).

Capitolo 7

Casi di studio

In questo capitolo sarà applicata la metodologia, precedentemente proposta, a due casi di studio. Successivamente i risultati ottenuti saranno analizzati al fine di capire se la suddetta metodologia è atta al testing delle applicazioni mobile di realtà aumentata.

7.1 Caso di studio 1: Safari Animal AR

Il primo progetto, su cui sarà applicato il test automation, è uno di quelli utilizzati per l'effettuazione dell'analisi presente nel capitolo 5. Il nome dell'applicazione è "Safari Animals AR" ed è scaricabile anche da Google Play Store. Tale applicazione è stata scelta poiché risulta essere conforme alle tecnologie target individuate. Questa è una prerogativa fondamentale, poiché non è sempre possibile aprire progetti Unity creati con una versione differente dello stesso. Questa problema-

tica si acuisce maggiormente con le applicazioni di realtà aumentata che utilizzano Vuforia, poiché il modo in cui esso viene importato varia in base alla versione dell'Engine. Altra caratteristica che ha spinto nella scelta di "Safari Animals AR" è stata la chiara spiegazione del funzionamento e degli script utilizzati, presenti nella pagina di GitHub raggiungibile al seguente link [23]. Nonostante ciò, in essa non è presente alcuna tipologia di test e per questa ragione si è deciso di testarne il funzionamento. L'applicazione è di tipo marker-based e il suo obiettivo è di tipo ludico. Una volta avviata, l'applicazione automaticamente setterà l'orientamento dello smartphone in orizzontale grazie allo script "AutoFocus", di cui si mostra il codice nella figura 7.1. Se il marker, mostrato in figura 7.12, sarà inquadrato, su di esso sarà creata un'ambientazione simil-Savana con al centro la zebra (animale presente di default) e, nella parte bassa dello schermo, compariranno sei bottoni e un pad. Ognuno dei sei bottoni è associato a uno specifico animale, cliccandoci comparirà l'animale ad esso associato e sarà l'unico visibile nella scena. Tale gestione dei bottoni è ottenuta mediante lo script "AnimalSpawn" mostrato in figura 7.2. Inoltre è possibile controllare il movimento dell'animale presente sulla scena trascinando il pad nella direzione desiderata, comportamento ottenuto con l'ultimo dei tre script presenti nel progetto, "AnimalsController", di cui si presenta il codice nella figura 7.3.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Vuforia;

public class AutoFocus : MonoBehaviour {

    void Start()
    {
        // Get the Vuforia instance of the app
        var vuforia = VuforiaARController.Instance;

        // Register OnVuforiaStarted event
        vuforia.RegisterVuforiaStartedCallback(OnVuforiaStarted);

        // Register OnPaused event
        vuforia.RegisterOnPauseCallback(OnPaused);
    }

    // As soon as Vuforia is initialized
    private void OnVuforiaStarted()
    {
        // Override FocusMode
        CameraDevice.Instance.SetFocusMode(

            // Enable continuos autofocus
            CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
    }

    // When Vuforia is paused
    private void OnPaused(bool paused)
    {
        // When it's resumed
        if (!paused)
        {
            // Focus mode may be reset to default
            // Therefore, re-enable autofocus
            CameraDevice.Instance.SetFocusMode(
                CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
        }
    }
}
```

Figura 7.1: Script Autofocus

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AnimalSpawn : MonoBehaviour {

    // Define a new list containing GameObjects
    private List<GameObject> models;

    // Variable holding the animal that has been selected from the bottom menu
    private int SelectionIndex = 0;

    private void Start () {

        // Create a new empty list from the above variable
        models = new List<GameObject>();

        // For each child of the GameObject this script has been applied to
        // i.e. all the animals under the "SafariAnimalsList" GameObject
        foreach(Transform t in transform)
        {
            // Add the animal to the "models" list
            models.Add(t.gameObject);

            // Set the animal's visibility to hidden
            // i.e. active = false
            t.gameObject.SetActive(false);
        }

        // Only show the animal that has been selected
        // i.e. upon launching the app, models[0] (which is the zebra) will be the only animal visible
        models[SelectionIndex].SetActive(true);
    }

    // Event detecting change of index
    // i.e. event triggered upon pressing one of the animals from the bottom menu (each entry is a button)
    public void Select(int index) {

        // If the selected animal is already displayed on screen
        if (index == SelectionIndex)

            // Nothing happens
            return;

        // If the value of index is different than the number of animals contained in the list
        // (in case the app is glitched)
        if (index < 0 || index >= models.Count)

            // Nothing happens
            return;

        // Otherwise, hide the currently displayed animal
        models[SelectionIndex].SetActive(false);

        // Change SelectionIndex to the newly selected animal
        SelectionIndex = index;

        // Display this animal
        models[SelectionIndex].SetActive(true);
    }
}
```

Figura 7.2: Script AnimalSpawn

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;

public class AnimalsController : MonoBehaviour
{
    // Define the RigidBody component
    private Rigidbody rb;

    // Define the Animation component
    private Animation anim;

    void Start()
    {
        // Assign the above variable "rb" to the GameObject's RigidBody component
        rb = GetComponent<Rigidbody>();

        // Assign the above variable "anim" to the GameObject's Animation component
        anim = GetComponent<Animation>();
    }

    // For each frame
    void Update()
    {
        // Get the horizontal axis from the joystick
        // (CrossPlatformInput is the library which controls the joystick)
        float x = CrossPlatformInputManager.GetAxis("Horizontal");

        // Get the vertical axis from the joystick
        float y = CrossPlatformInputManager.GetAxis("Vertical");

        // Create a new vector, which will hold the position of the joystick
        Vector3 movement = new Vector3(x, 0, y);

        // Rate of speed the animals will be moving
        rb.velocity = movement * 0.15f;

        // If joystick's x and y values are different from 0
        // i.e. joystick has been moved
        if (x != 0 && y != 0)
        {
            // Move the animal in the direction of the joystick
            transform.eulerAngles = new Vector3(transform.eulerAngles.x, Mathf.Atan2(x, y) * Mathf.Rad2Deg, transform.eulerAngles.z);
        }

        // If either x or y values are different from 0
        // i.e. animal is moving
        if (x != 0 || y != 0)
        {
            // Play the walk animation
            anim.Play("walk");
        }

        // Otherwise
        // i.e. animal is not being moved
        else
        {
            // Play the idle animation
            anim.Play("idle");
        }
    }
```

Figura 7.3: Script AnimalsController

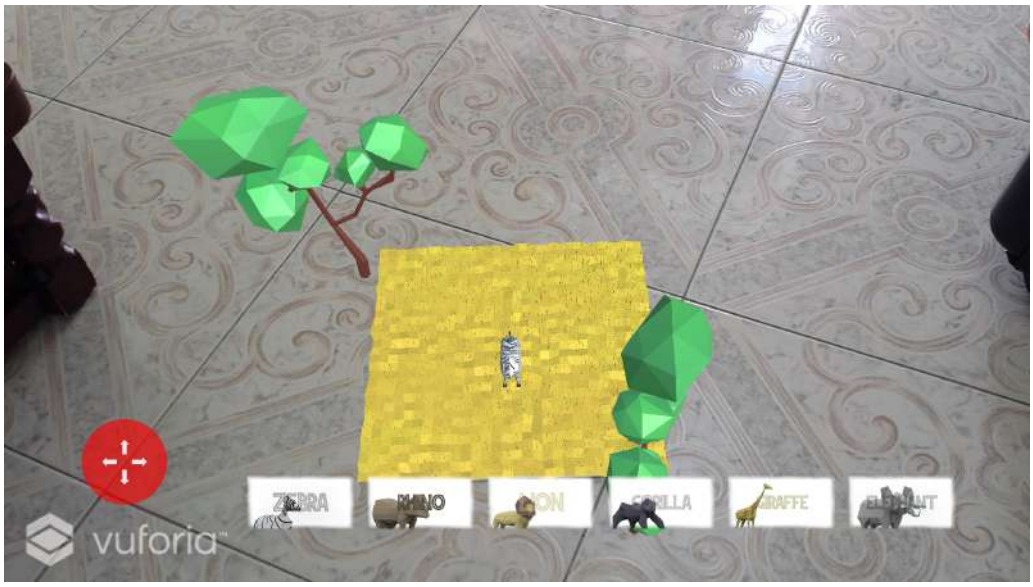


Figura 7.4: GUI applicazione

7.1.1 Import dell'applicazione

Innanzi tutto si è scaricato da GitHub il file ".zip" contenente il progetto per poi estrarlo e importarlo in UnityHub selezionando l'apposita versione di Unity (2018.4.30f1). Dopo aver aperto il progetto con Unity, è stato necessario inserire la chiave personale per attivare le funzioni di Vuforia. Successivamente si è cambiata l'immagine utilizzata come marker. Ciò è stato necessario perché quella linkata dal realizzatore dell'applicazione non era presente nel link fornito. Questo cambiamento non ha apportato alcuna modifica al funzionamento generale dell'applicazione, l'unica differenza consta esclusivamente nella diversa immagine utilizzata come marker. L'immagine scelta è una di quelle messe a disposizione di default dal database delle immagini di Vuforia 7.12. In seguito, è stato importato Poco-SDK all'interno del progetto

effettuando i passi mostrati nel paragrafo 4.5.3. In questo modo, sarà possibile accedere alla gerarchia di oggetti per effettuare un'implementazione più precisa dello script di test. Invece per quanto riguarda la creazione del file di log, necessario per la valutazione della copertura dei rami, al progetto è stato aggiunto lo script "SondaManager", mostrato nella sezione 6.3.1, e a ognuno degli script sono state aggiunte le sonde seguendo la metodologia proposta, anch'essa, nella sezione 6.3.1. Di seguito sono mostrati gli script del progetto che, rispetto a quelli mostrati nelle figure 7.1, 7.2 e 7.3, presentano l'aggiunta delle suddette sonde. In fine, l'applicazione è stata deployata sul dispositivo emulato mostrato nei capitoli precedenti.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Vuforia;

public class AutoFocus : MonoBehaviour {


    void Start()
    {
        SondaManager.inserisciSonda("AutoFocus.start");

        // Get the Vuforia instance of the app
        var vuforia = VuforiaARController.Instance;

        // Register OnVuforiaStarted event
        vuforia.RegisterVuforiaStartedCallback(OnVuforiaStarted);

        // Register OnPaused event
        vuforia.RegisterOnPauseCallback(OnPaused);
    }

    // As soon as Vuforia is initialized
    private void OnVuforiaStarted()
    {
        SondaManager.inserisciSonda("AutoFocus.OnVuforiaStarted");
        // Override FocusMode
        CameraDevice.Instance.SetFocusMode(

            // Enable continuos autofocus
            CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
    }

    // When Vuforia is paused
    private void OnPaused(bool paused)
    {
        SondaManager.inserisciSonda("AutoFocus.OnPaused");
        // When it's resumed
        if (!paused)
        {
            SondaManager.inserisciSonda("AutoFocus.OnPaused.if (!paused)");
            // Focus mode may be reset to default
            // Therefore, re-enable autofocus
            CameraDevice.Instance.SetFocusMode(
                CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
        }
    }
}
```

Figura 7.5: Script AutoFocus con sonde

```
// Define a new list containing GameObjects
private List<GameObject> models;

// Variable holding the animal that has been selected from the bottom menu
private int SelectionIndex = 0;


private void Start () {

    SondaManager.inserisciSonda("AnimalSpawn.start");

    // Create a new empty list from the above variable
    models = new List<GameObject>();

    // For each child of the GameObject this script has been applied to
    // i.e. all the animals under the "SafariAnimalsList" GameObject
    foreach(Transform t in transform)
    {
        SondaManager.inserisciSonda("AnimalSpawn.start.foreach(Transform t in transform)");
        // Add the animal to the "models" list
        models.Add(t.gameObject);

        // Set the animal's visibility to hidden
        // i.e. active = false
        t.gameObject.SetActive(false);
    }

    // Only show the animal that has been selected
    // i.e. upon launching the app, models[0] (which is the zebra) will be the only animal visible
    models[SelectionIndex].SetActive(true);
}

// Event detecting change of index
// i.e. event triggered upon pressing one of the animals from the bottom menu (each entry is a button)
public void Select(int index) {

    SondaManager.inserisciSonda("AnimalSpawn.Select");
    // If the selected animal is already displayed on screen
    if (index == SelectionIndex)
        {
            SondaManager.inserisciSonda("AnimalSpawn.Select.if (index == SelectionIndex)");
            // Nothing happens
            return;
        }



    // If the value of index is different than the number of animals contained in the list
    // (in case the app is glitched)
    if (index < 0 || index >= models.Count)
    {
        SondaManager.inserisciSonda("AnimalSpawn.Select.if (index < 0 || index >= models.Count)");
        // Nothing happens
        return;
    }


    // Otherwise, hide the currently displayed animal
    models[SelectionIndex].SetActive(false);

    // Change SelectionIndex to the newly selected animal
    SelectionIndex = index;

    // Display this animal
    models[SelectionIndex].SetActive(true);
}
}
```

Figura 7.6: Script AnimalSpawn con sonde

```
public class AnimalsController : MonoBehaviour
{
    // Define the RigidBody component
    private Rigidbody rb;

    // Define the Animation component
    private Animation anim;


    void Start()
    {

        // Assign the above variable "rb" to the GameObject's RigidBody component
        rb = GetComponent<Rigidbody>();

        // Assign the above variable "anim" to the GameObject's Animation component
        anim = GetComponent<Animation>();
        SondaManager.inserisciSonda("AnimalsController.start");

    }

    // For each frame
    void Update()
    {
        SondaManager.inserisciSonda("AnimalsController.update");
        // Get the horizontal axis from the joystick
        // (CrossPlatformInput is the library which controls the joystick)
        float x = CrossPlatformInputManager.GetAxis("Horizontal");

        // Get the vertical axis from the joystick
        float y = CrossPlatformInputManager.GetAxis("Vertical");

        // Create a new vector, which will hold the position of the joystick
        Vector3 movement = new Vector3(x, 0, y);

        // Rate of speed the animals will be moving
        rb.velocity = movement * 0.15f;

        // If joystick's x and y values are different from 0
        // i.e. joystick has been moved
        if (x != 0 && y != 0)
        {
            // Move the animal in the direction of the joystick
            transform.eulerAngles = new Vector3(transform.eulerAngles.x, Mathf.Atan2(x, y) * Mathf.Rad2Deg, transform.eulerAngles.z);
            SondaManager.inserisciSonda("AnimalsController.update.if (x != 0 && y != 0)");
        }

        // If either x or y values are different from 0
        // i.e. animal is moving
        if (x != 0 || y != 0)
        {
            // Play the walk animation
            anim.Play("walk");
            SondaManager.inserisciSonda("AnimalsController.update.if (x != 0 || y != 0)");
        }

        // Otherwise
        // i.e. animal is not being moved
        else
        {
            // Play the idle animation
            anim.Play("idle");
            SondaManager.inserisciSonda("AnimalsController.update.if (x != 0 || y != 0).else");
        }
    }
}
```

Figura 7.7: Script AnimalsController con sonde

7.1.2 Modellazione

In base a quanto detto nella sezione 6.2.4, si procede con la modellazione dell'applicazione scelta. Nella fattispecie, "Safari Animal AR" non

presenta un vero e proprio SRS, ma le informazioni relative al suo funzionamento possono essere ricavate dalla descrizione presente sul sito GitHub [23] e su Google Play Store, le quali sono state utilizzate per la stesura del paragrafo 7.1 utile a chiarirne il funzionamento. Di seguito saranno applicate le nozioni espresse nella sezione 6.2.2 per modellare la FSM dell'applicazione in esame. Innanzi tutto è necessario capire quali siano gli event di tale applicazione, nella tabella 7.1 sono definiti gli event, estrapolati dalla descrizione di "Safari Animal AR", con le conseguenti action.

Event	**Action**
Riconoscimento marker	Creazione scena Savana con zebra
Disconoscimento marker	Scomparsa scena savana
Click bottone zebra	Comparsa zebra sulla scena
Click bottone rinoceronte	Comparsa rinoceronte sulla scena
Click bottone leone	Comparsa leone sulla scena
Click bottone gorilla	Comparsa gorilla sulla scena
Click bottone giraffa	Comparsa giraffa sulla scena
Click bottone elefante	Comparsa elefante sulla scena
Spostamento pad su	Movimento animale in su
Spostamento pad giù	Movimento animale in giù
Spostamento pad a destra	Movimento animale a destra
Spostamento pad a sinistra	Movimento animale a sinistra

Tabella 7.1: Tabella event e action

Si passa a definire il numero di stati necessari a descrivere tale applicazione. Possiamo subito definire due macro-stati: uno in cui non è stato riconosciuto il marker e l'altro in cui il marker è stato riconosciuto e, grazie a ciò, viene creata la scena di realtà aumentata contenente gli event presentati in tabella, fatta esclusione per i primi due relativi al riconoscimento e disconoscimento del marker. Di seguito si mostra la suddetta modellazione da raffinare.

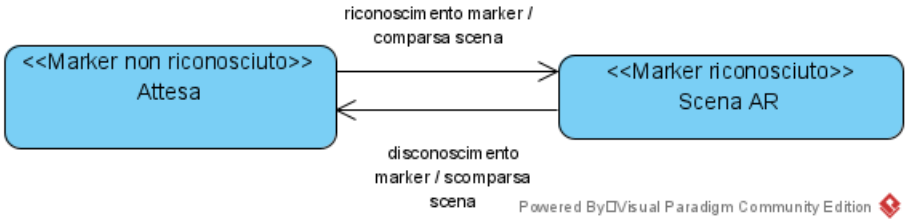


Figura 7.8: FSM con macro-stati

E' necessario effettuare il raffinamento dello stato "Scena AR" e, fatta eccezione per le prime due righe della tabella, a una prima occhiata si potrebbe pensare di associare uno stato a ciascuna action. Per capire se tale affermazione risulta essere corretta è necessario applicare la nozione di invariante di stato e post-condizioni. Per quanto riguarda l'invariante di stato si hanno le seguenti condizioni:

- verificare la comparsa della scena con l'animale;
- verificare l'assenza della scena;
- verificare l'avvenuto movimento dell'animale.

Per quanto riguarda le post-condizioni si hanno le situazioni espresse nella colonna delle action. Si può quindi intuire che la comparsa di

diverse tipologie di animali non implica la creazione di un nuovo stato ma semplicemente è possibile associarle alla condizione che sia comparsa la scena con l'animale, eventualmente scelto in seguito al click del bottone. Quindi in definitiva gli stati riconoscibili nell'applicazione in esame sono:

- Attesa: verifica l'assenza della scena AR.
- Scena con animale: verifica che la scena con l'animale, eventualmente scelto da uno dei bottoni, sia presente;
- Animale mosso: verifica l'avvenuto movimento dell'animale.

Determinati gli stati si passa alla definizione delle transition applicando l'omonima nozione, descritta nella sezione 6.2.2. Tale nozione afferma che a ogni event deve essere associata una transition, di conseguenza a ogni event presente in tabella 7.1 è associata la transition necessaria allo spostamento tra gli stati individuati. La FSM che ne scaturisce è mostrata di seguito.

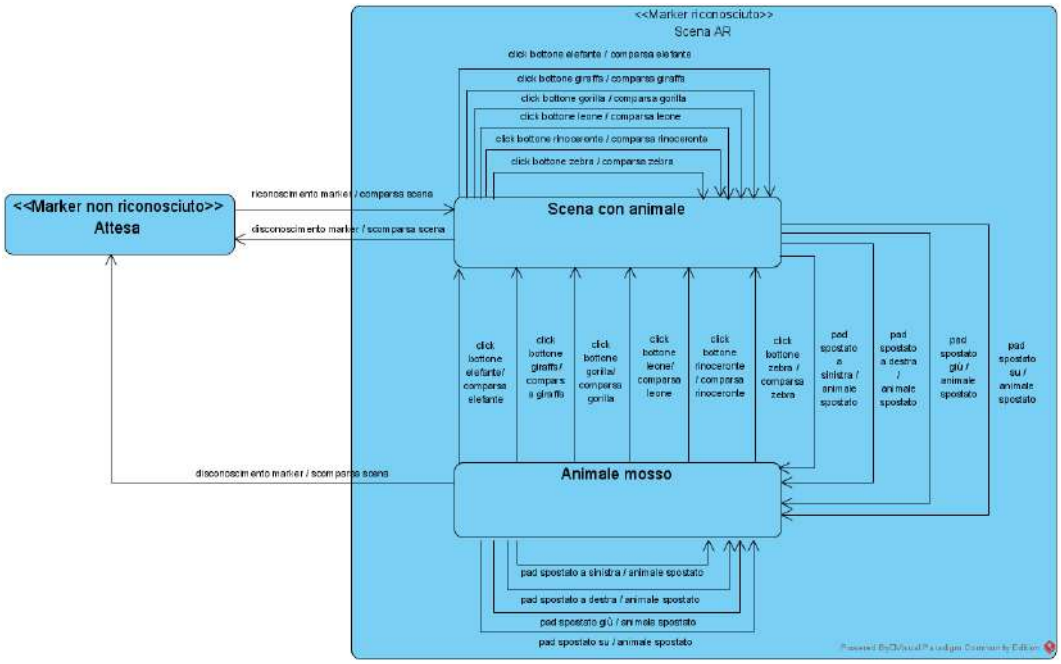


Figura 7.9: FSM derivata

Dall'immagine si può evincere che a ogni event di click del bottone e di movimento del pad sono associate due transition, mentre una ciascuno per il riconoscimento e disconoscimento del marker. Si noti che nessuna delle transizioni presenta condizioni di guardia dato che non vi è alcun event che, all'interno del medesimo stato, può generare una transizione verso due o più stati differenti. Avendo dimostrato che nessuna delle transizioni presenta condizioni di guardia, il modello può essere semplificato, mediante l'utilizzo di archi parametrizzati, per questioni di leggibilità. Di seguito si mostra la FSM risultante.

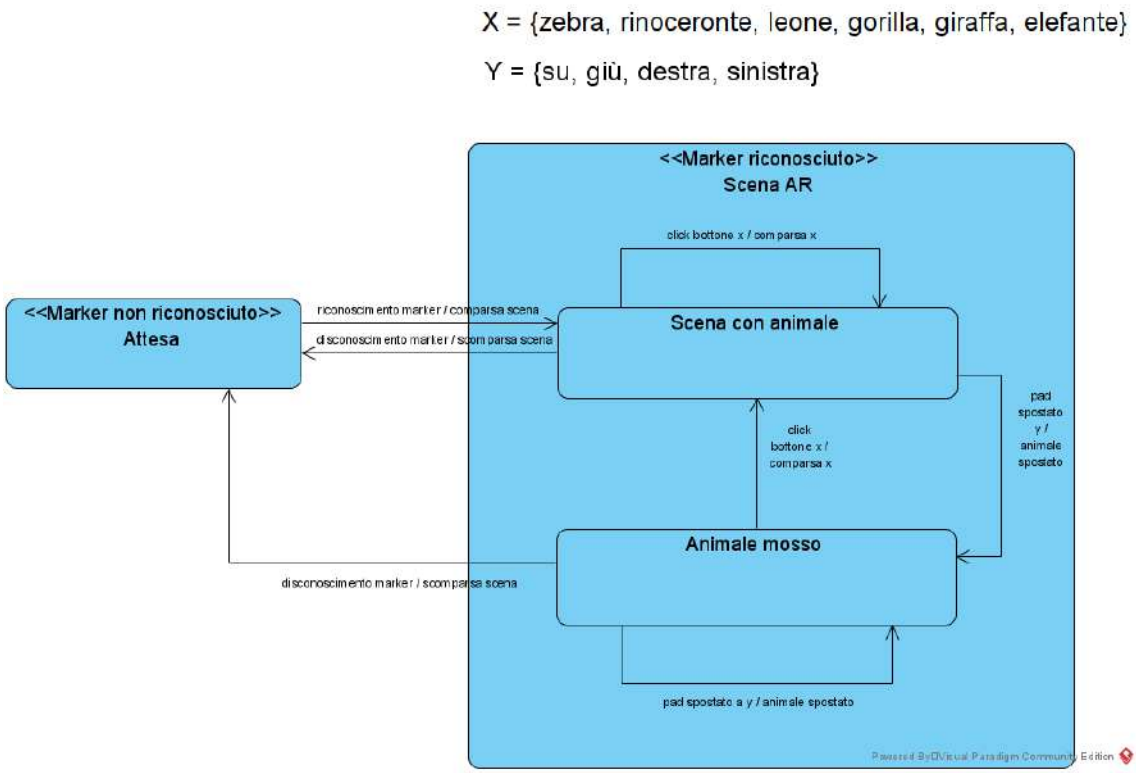


Figura 7.10: FSM derivata parametrizzata

Nella figura soprastante sono state effettuate le seguenti semplificazioni:

- I sei cappi di "Scena con animale" sono stati sostituiti da un unico cappio parametrizzato, dove x rappresenta la variabile appartenente al dominio X, discreto e finito, contenete le diverse tipologie di animali. Chiaramente, ogni volta che si sceglie di utilizzare tale transizione, la variabile x può assumere solo uno dei valori del dominio X;

- Le quattro transizioni da "Scena con Animale" verso "Animale mosso", sono state sostituite da un'unica transizione parametrizzata, dove y rappresenta la variabile appartenente al dominio Y,

discreto e finito, contenente le diverse direzioni di spostamento. Chiaramente, ogni volta che si sceglie di utilizzare tale transizione, la variabile y può assumere solo uno dei valori del dominio Y;

- I quattro cappi di "Animale mosso" sono stati sostituiti da un unico cappio parametrizzato mediante l'utilizzo della variabile $y \in Y$, precedentemente definita.

- Le sei transizioni da "Animale mosso" verso "Scena con animale" sono state sostituite da un'unica transizione parametrizzata mediante l'utilizzo della variabile $x \in X$, precedentemente definita.

Tuttavia la FSM presentata in figura 7.10 non soddisfa la nozione relativa allo stato iniziale e quello finale. Ciò significa che non è presente alcuno stato avente solo transizioni uscenti (stato iniziale), né alcuno avente solo transizioni in ingresso (stato finale). Quindi al modello proposto vanno inseriti gli pseudo-stati iniziale e finale con le relative transizioni. Nel caso in esame vi sarà una sola transizione che parte stato iniziale e giunge allo stato di "Attesa" e consisterà nell'event di avvio dell'applicazione (omesso nella tabella 7.1); mentre lo stato finale avrà tre transizioni in ingresso aventi origine rispettivamente da "Attesa", "Scena con animale" e "Animale mosso" e rappresentanti

la chiusura dell'applicazione (anch'essa omessa nella tabella 7.1). Di seguito è mostrata la FSM risultante.

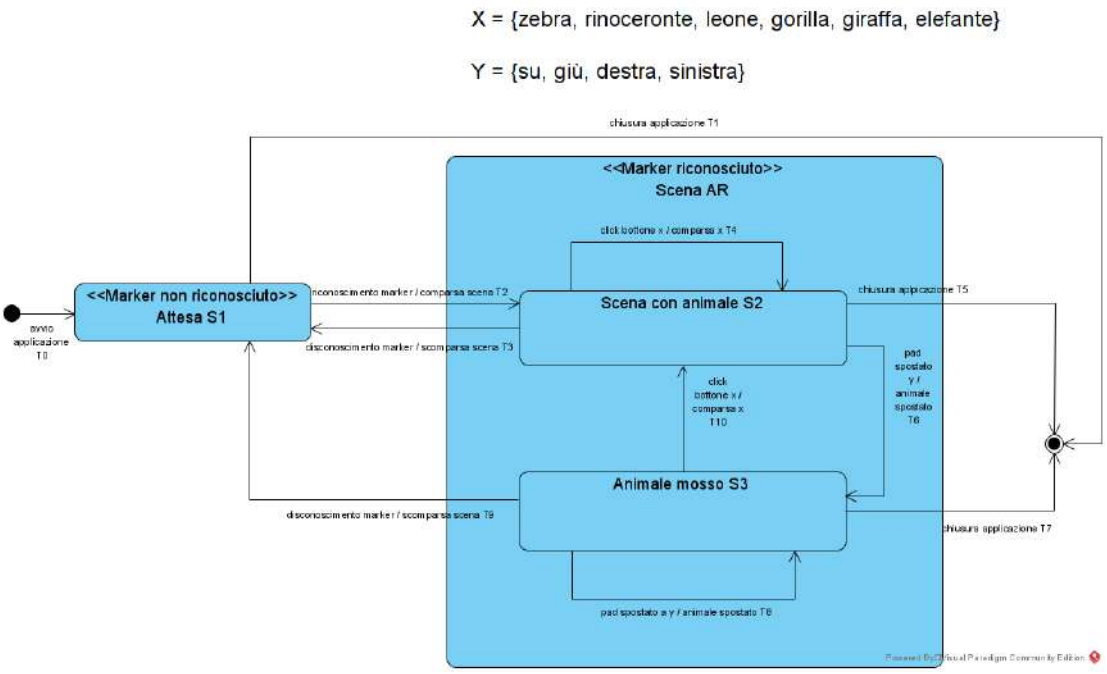


Figura 7.11: FSM completa

Si passa ora alla verifica della Reachability, ossia la verifica della raggiungibilità di ogni stato. Per la accertarsi che tale nozione sia soddisfatta, si verifica che ogni stato, ad esclusione di quello iniziale e finale, abbia almeno una transizione in ingresso e una in uscita. Come si può evincere dall'immagine sopra riportata, ciò è verificato e di conseguenza è scongiurata la presenza di Dead state, Dead loop e Magic state. Avendo ora il modello completo della FSM è necessario validare che tale modello sia minimo, ossia non presenti stati equivalenti. Per accertarsi di ciò è necessario verificare che tutte le sequenze di event possibili di due o più stati non producano lo stesso comportamento. Nel caso in esame ciò è facilmente verificabile dato che sono presenti

tre stati, escludendo gli pseudo-stati di inizio e fine. Di fatto le coppie di stati da analizzare risultano: "Attesa"-"Scena con animale" e "Scena con animale"-"Animale mosso", perché la coppia "Attesa"-"Animale mosso" possiede una sola transizione da "Animale mosso" ad Attesa" di disconoscimento del marker. Riguardo la prima coppia interagisce con lo stato "Scena con animale" attraverso le due transizioni di avvenuto riconoscimento e disconoscimento del marker. Tali transizioni portano rispettivamente da "Attesa" a "Scena con animale" e viceversa, quindi si può concludere che i due stati coinvolti non producono lo stesso comportamento. Per la coppia "Scena con animale"-"Animale mosso" può essere fatto un ragionamento analogo. Tra essi sono presenti le transizione di click del bottone e di movimento del pad, tuttavia in base allo stato in cui ci si trova i comportamenti prodotti sono differenti. Infatti nel caso in cui ci si trovi nello stato "Scena con animale" le transizioni relative al click del bottone sono cappi e non permettono l'uscita da tale stato, mentre le transizioni di movimento del pad conducono nello stato "Animale mosso". Quando ci si trova nello stato "Animale mosso" succede l'esatto opposto, ossia le transizioni relative al movimento del pad risultano cappi che non permettono l'uscita da tale stato, mentre le transizioni relative al click dei bottoni portano nello stato "Scena con animale". Da quanto detto si può sancire che i due stati non sono equivalenti. Inoltre, essendo che nessuno dei tre stati è equivalente a un altro, si può stabilire che la FSM di figura 7.11

è minima. Si noti che tale FSM presenta gli stati e le transizioni numerate, per meglio capire a quali di essi ci si riferirà in ogni test case. In conclusione, avendo verificato tutte le nozioni sul modello creato, da esso è possibile scaturire le suite di test che dovranno essere effettuate sull'applicazione.

7.1.3 Definizione della test suite

Avendo a disposizione il modello creato nella sezione precedente è possibile scaturire da esso la test suite, utilizzando uno dei criteri di copertura mostrati nella sezione 6.2.3. In base al criterio che sarà scelto varieranno gli obiettivi e il numero di test da effettuare. Tra i criteri di copertura proposti quello che sarà scartato a prescindere è All-Paths Coverage poiché, essendo che la FSM modellata presenta cicli e cappi, il numero di path possibili risulterebbe infinito. Di seguito saranno applicati i restanti criteri di copertura, in modo tale da constatare quale fra essi risulta il migliore per il testing dell'applicazione in esame. Per ognuno dei criteri selezionati sarà creata una tabella della test suite, presentata nella sezione 6.2.5, contenete i test case da effettuare. Si noti che tra i vari test case, presenti nelle successive test suite, nessuno è atto a verificare la correttezza del testo presente sui bottoni. Ciò è dovuto al fatto che ad essi è stato inserito un componente di tipo "Image" il cui scopo è "rivestirli" con la scritta e una miniatura del relativo animale. In altre parole ciò che si vede sul bottone non è un testo ma

bensì un'immagine e come tale non è possibile, con questi strumenti, verificarne la correttezza.

Test suite di All-States Coverage

L'obiettivo di tale criterio di copertura, come si può intuire dal nome, consiste nel coprire tutti gli stati almeno una volta. Quindi ciò che è necessario fare è individuare i paths che permettano di raggiungere tale obiettivo. Anche in questo caso i paths possibili risulterebbero infiniti a causa della presenza di cappi e cicli. Pertanto ci si limiterà a soddisfare il requisito minimo di tale criterio attraversando ogni stato al più una volta per ogni path. Tralasciando il pseudo-stato iniziale, lo stato di partenza dell'applicazione sarà "Attesa" avente una sola transizione in uscita che permette il raggiungimento dello stato "Scena con animale". Una volta passati in questo stato si tralasciano i cappi e la transizione di disconoscimento del marker, perché porterebbero rispettivamente nello stato di "Scena con animale" e nello stato di "Attesa" entrambi già visitati. Di conseguenza le uniche transizioni che porterebbero in un nuovo stato sono quelle relative al movimento del pad, che giungono allo stato "Animale mosso". Da quest'ultimo stato, tralasciando la transizione verso il pseudo-stato finale, vi sono esclusivamente transizioni che portano: verso "Attesa", verso "Scena con animale" o verso il medesimo stato e per questo, qualsiasi transizione si scelga, si ritornerebbe in uno stato già visitato. Pertanto, i possibili

paths per coprire tutti gli stati risultano essere quattro e sono differenziati dalla transizione di spostamento del pad. Nelle tabelle 7.2, 7.3, 7.4 e 7.5 si mostra la test suite risultante dal criterio selezionato.

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
1	Comparsa della scena safari	Inquadramento marker	Comparsa degli oggetti che compongono la scena e animale di default (zebra)	S1, S2	T2
2	Trascinamento del pad verso giù	Presenza della scena e animale di default (zebra)	Avvenuto spostamento dell'animale verso il basso rispetto al punto in cui si trovava	S2, S3	T6 (y = giù)

Tabella 7.2: Tabella test suite All-States Coverage path 1

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
1	Comparsa della scena safari	Inquadramento marker	Comparsa degli oggetti che compongono la scena e animale di default (zebra)	S1, S2	T2

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
3	Trascinamento del pad verso su	Presenza della scena e animale di default (zebra)	Avvenuto spostamento dell'animale verso l'alto rispetto al punto in cui si trovava	S2, S3	T6 (y = su)

Tabella 7.3: Tabella test suite All-States Coverage path 2

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
1	Comparsa della scena safari	Inquadramento marker	Comparsa degli oggetti che compongono la scena e animale di default (zebra)	S1, S2	T2
4	Trascinamento del pad verso destra	Presenza della scena e animale di default (zebra)	Avvenuto spostamento dell'animale verso destra rispetto al punto in cui si trovava	S2, S3	T6 (y = destra)

Tabella 7.4: Tabella test suite All-States Coverage path 3

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
1	Comparsa della scena safari	Inquadramento marker	Comparsa degli oggetti che compongono la scena e animale di default (zebra)	S1, S2	T2
5	Trascinamento del pad verso sinistra	Presenza della scena e animale di default (zebra)	Avvenuto spostamento dell'animale verso sinistra rispetto al punto in cui si trovava	S2, S3	T6 (y = sinistra)

Tabella 7.5: Tabella test suite All-States Coverage path 4

Si noti che il test avente ID pari a 1 sarà utilizzato in tutti i paths poiché con esso si effettua il passaggio dallo stato "Attesa" a "Scena con animale". Inoltre si fa presente che per coprire tutti gli stati è necessario scegliere solo uno dei path trovati.

	Coperti	Totali	Percentuale
Stati	3	3	100 %
Transizioni	2	23	8,7 %

Tabella 7.6: Percentuale copertura Stati e Transizioni All-States Coverage

In tabella 7.6 si riporta la statistica relativa alla valutazione della

percentuale di copertura degli stati e delle transizioni. Si noti che il numero di transizioni totali è dato dalla somma di tutte le possibili declinazioni di ogni transizione parametrizzata. A tale somma va aggiunta la transizione di riconoscimento e le due di disconoscimento del marker. La tabella di cui sopra mostra che, con tale test suite, sono stati coperti tutti i possibili stati ma solo una piccola percentuale di transizioni.

N° test case
2

Tabella 7.7: N° test case All-States Coverage

La tabella 7.7 definisce il numero di test che si dovranno effettuare scegliendo uno dei paths.

Test suite di All-Transitions Coverage

Lo scopo che si pone tale criterio è coprire tutte le possibili transizioni almeno una volta. Di conseguenza la test suite scaturente può definirsi completa se ogni transizione è stata coperta almeno una volta. Tralasciando il pseudo-stato iniziale e finale, a cui sono associate le transizioni di avvio e chiusura dell'applicazione, per ognuno dei rimanenti stati sarà necessario verificare le transizioni ad esso associate. Di seguito si mostra la test suite relativa a tale criterio.

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
1	Comparsa della scena Safari e animale di default	Inquadramento marker	Comparsa degli oggetti che compongono la scena e animale di default (zebra)	S1, S2	T2
2	Click del bottone relativo alla zebra	Presenza della scena e di un animale	Comparsa della zebra sulla scena in seguito alla sostituzione del precedente animale	S2	T4 (x = zebra)
3	Click del bottone relativo al rinoceronte	Presenza della scena e di un animale	Comparsa del rinoceronte sulla scena in seguito alla sostituzione del precedente animale	S2	T4 (x = rinoceronte)
4	Click del bottone relativo al leone	Presenza della scena e di un animale	Comparsa del leone sulla scena in seguito alla sostituzione del precedente animale	S2	T4 (x = leone)

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
5	Click del bottone relativo al gorilla	Presenza della scena e di un animale	Comparsa del gorilla sulla scena in seguito alla sostituzione del precedente animale	S2	T4 (x = gorilla)
6	Click del bottone relativo alla giraffa	Presenza della scena e di un animale	Comparsa della giraffa sulla scena in seguito alla sostituzione del precedente animale	S2	T4 (x = giraffa)
7	Click del bottone relativo all'elefante	Presenza della scena e di un animale	Comparsa dell'elefante sulla scena in seguito alla sostituzione del precedente animale	S2	T4 (x = elefante)
8	Click del bottone relativo alla zebra	Animale precedentemente spostato	Comparsa della zebra sulla scena in seguito alla sostituzione del precedente animale	S3, S2	T4 (x = zebra)

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
9	Click del bottone relativo al rinoceronte	Animale precedentemente spostato	Comparsa del rinoceronte sulla scena in seguito alla sostituzione del precedente animale	S3, S2	T4 (x = rinoceronte)
10	Click del bottone relativo al leone	Animale precedentemente spostato	Comparsa del leone sulla scena in seguito alla sostituzione del precedente animale	S3, S2	T4 (x = leone)
11	Click del bottone relativo al gorilla	Animale precedentemente spostato	Comparsa del gorilla sulla scena in seguito alla sostituzione del precedente animale	S3, S2	T4 (x = gorilla)
12	Click del bottone relativo alla giraffa	Animale precedentemente spostato	Comparsa della giraffa sulla scena in seguito alla sostituzione del precedente animale	S3, S2	T4 (x = giraffa)

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
13	Click del bottone relativo all'elefante	Animale precedentemente spostato	Comparsa dell'elefante sulla scena in seguito alla sostituzione del precedente animale	S3, S2	T4 (x = elefante)
14	Trascinamento del pad verso sopra	Presenza della scena e di un animale	Spostamento dell'animale in su rispetto al punto in cui si trovava	S2, S3	T6 (y = su)
15	Trascinamento del pad verso giù	Presenza della scena e di un animale	Spostamento dell'animale in giù rispetto al punto in cui si trovava	S2, S3	T6 (y = giù)
16	Trascinamento del pad verso destra	Presenza della scena e di un animale	Spostamento dell'animale a destra rispetto al punto in cui si trovava	S2, S3	T6 (y = destra)
17	Trascinamento del pad verso sinistra	Presenza della scena e di un animale	Spostamento dell'animale a sinistra rispetto al punto in cui si trovava	S2, S3	T6 (y = sinistra)

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
18	Trascinamento del pad verso sopra	Animale precedentemente spostato	Spostamento dell'animale in su rispetto al punto in cui si trovava	S3	T6 (y = su)
19	Trascinamento del pad verso giù	Animale precedentemente spostato	Spostamento dell'animale in giù rispetto al punto in cui si trovava	S3	T6 (y = giù)
20	Trascinamento del pad verso destra	Animale precedentemente spostato	Spostamento dell'animale a destra rispetto al punto in cui si trovava	S3	T6 (y = destra)
21	Trascinamento del pad verso sinistra	Animale precedentemente spostato	Spostamento dell'animale a sinistra rispetto al punto in cui si trovava	S3	T6 (y = sinistra)
22	Scomparsa della scena Safari e animale di default in seguito al disconoscimento del marker	Presenza della scena e di un animale	Scomparsa della scena Safari e animale di default con conseguente attesa di ricomparsa marker	S2, S1	T3

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
23	Scomparsa della scena Safari e animale di default in seguito al disconoscimento del marker	Animale precedentemente spostato	Scomparsa della scena Safari e animale di default con conseguente attesa di ricomparsa marker	S3, S1	T9

Tabella 7.8: Tabella test suite All-Transitions Coverage

Si noti che le operazioni di click e trascinamento del pad sono ripetute due volte, poichè vanno a coprire i cappi e le transizioni aventi stato sorgente e destinazione diversi. E' facile intuire che eseguendo tale test suite saranno visitati tutti gli altri stati anche più di una volta. Per confermare quanto detto, si riporta la tabella 7.9 che mostra le percentuali di copertura riguardanti stati e transizioni.

	Coperti	Totali	Percentuale
Stati	3	3	100 %
Transizioni	23	23	100 %

Tabella 7.9: Percentuale copertura Stati e Transizioni All-Transitions Coverage

Di seguito si presenta la tabella contenente il numero di test case da effettuare eseguendo tale test suite.

N° test case
23

Tabella 7.10: N° test case All-Transitions Coverage

All-One-Loop-Paths Coverage

Lo scopo che si pone tale criterio è individuare tutti i possibili paths contenenti uno stato ripetuto. Si può subito intuire che tali paths potrebbero avere una lunghezza differente. Per lunghezza s'intende il numero di stati, ammettendo quelli ripetuti, visitati nello specifico path. Ad esempio considerando il path "Attesa"-"Scena con animale"-"Attesa", esso risulta essere valido e avere lunghezza tre poiché lo stato "Attesa" è presente due volte. Chiarito questo concetto si passa al calcolo di tutti i possibili paths validi, tralasciando gli pseudo-stati iniziale e finale. Si fa presente che, al fine di avere uno stato ripetuto, la lunghezza dei paths deve essere almeno pari a tre. Si procede con il calcolo di tali percorsi. I primi due stati devono necessariamente essere ordinati nel seguente modo: "Attesa" e "Scena con animale". Ciò è dovuto dal modello della FSM dell'applicazione che impone il suddetto ordine. Dallo stato "Scena con animale" è possibile ottenere dei paths validi di lunghezza tre in due modi: utilizzando la transizione di disconoscimento del marker per tornare allo stato "Attesa" oppure prendendo ognuno dei sei valori del cappio parametrizzato che giungono nel medesimo stato. Di conseguenza i paths di lunghezza 3 validi

risultano essere sette, infatti percorrendo la transizione parametrizzata di movimento del pad passeremmo allo stato "Animale mosso", creando paths non validi a causa dell'assenza di ripetizioni di stato. Si passa alla valutazione di paths validi di lunghezza quattro, le cui due prime posizione sono occupate rispettivamente da "Attesa" e "Scena con animale". Possono essere subito individuati quelli che scaturiscono dai sei paths aventi la ripetizione dello stato "Scena con animale". Infatti aggiungendo ad essi lo stato "Animale mosso" si ottengono paths validi. Tuttavia tale stato è raggiungibile mediante i quattro valori della transizione parametrizzata di movimento del pad, ne consegue che il numero di paths generati è pari a 24 (6 paths con ripetizione "Scena con animale" moltiplicati per le 4 transizioni, una per ogni valore della transizione parametrizzata, necessarie al raggiungimento dello stato "Animale mosso"). Oltre a questi possono esserne definiti altri, in particolare quelli aventi la ripetizione dello stato "Animale mosso" dato che tale stato presenta il cappio parametrizzato sfruttabile per la creazione di paths valdi. Per calcolare il numero di questi possibili paths si applica un ragionamento simile al precedente che porta al seguente risultato: 16 (dato dal prodotto delle 4 possibili transizioni, una per ogni valore della transizione parametrizzata, che portano da "Scena con animale" a "Animale mosso" per i 4 cappi, uno per ogni valore del cappio parametrizzato, di "Animale mosso"). Tuttavia da "Animale mosso" è possibile estrapolare altri paths validi, poiché esso

presenta ulteriori transizioni verso "Scena con animale" e "Attesa". Di conseguenza i paths che scaturiscono da tali transizioni sono rispettivamente: 24 aventi "Scena con animale" ripetuto (dato dal prodotto delle 4 possibili transizioni, una per ogni valore della transizione parametrizzata, che portano da "Scena con animale" a "Animale mosso" per le 6 possibili transizioni, una per ogni valore della transizione parametrizzata, che portano da "Animale mosso" a "Scena con animale") e 4 aventi "Attesa" ripetuto (dato dal prodotto delle 4 possibili transizioni, una per ogni valore della transizione parametrizzata, che portano da "Scena con animale" a "Animale mosso" per l'unica possibile che porta da "Animale mosso" ad "Attesa"). In definitiva il numero complessivo di path ottenuti è 75. Un numero estremamente alto se si considera che bisogna effettuare un test su ognuno degli stati di cui è composto un path. La ragione per cui non si riporta la test suite è che, nella fattispecie di "Safari Animal AR", tale test sarebbe superfluo in quanto i paths identificati non sono associati a particolari funzioni dell'applicazione attivabili esclusivamente dallo specifico path. Per questa ragione, tutti i paths trovati non farebbero altro che coprire più e più volte le transizioni presenti nel modello proposto, definendo, di fatto, una test suite simile a quella del criterio All-Transitions Coverage ma ridondante. Per confermare tale ipotesi nella successiva sezione, dopo aver implementato lo script di test e averlo eseguito, si confronteranno i risultati di questa test suite con quella scaturita dal

criterio All-Transition Coverage. Infine in tabella 7.11 sono riportate le percentuali di copertura di stati e transizioni.

	Coperti	**Totali**	**Percentuale**
Stati	3	3	100 %
Transizioni	23	23	100 %

Tabella 7.11: Percentuale copertura Stati e Transizioni All-One-Loop-Paths Coverage

Si noti che, anche se non è stata riportata la tabella della test suite di tale criterio, è facile ottenere questi numeri semplicemente analizzando i percorsi precedentemente definiti. Di seguito si mostra la tabella contenete il numero di test case da eseguire per tale test suite.

N° test case
218

Tabella 7.12: N° test case All-One-Loop-Paths Coverage

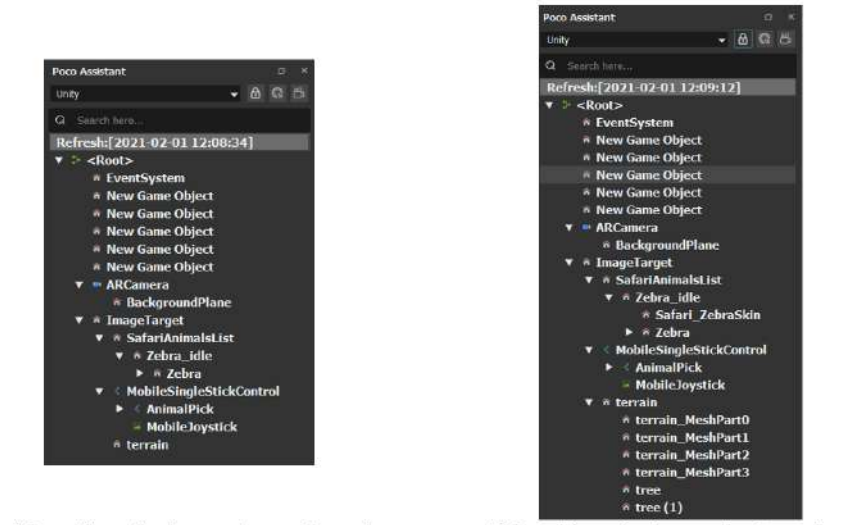
7.1.4 Implementazione ed esecuzione script di test

Il primo passo effettuato è stato avviare il dispositivo emulato su cui era presente "Safari Animal AR", successivamente si è lanciato AirtTestIDE e si è collegato il device emulato tramite ADB. Preventivamente, nel dispositivo, è stato posizionato il marker dell'applicazione sul tavolo della stanza. Una condizione che bisogna rispettare, per effettuare

i tests, è di inquadrare il marker nella giusta orientazione, come mostrato nell'immagine seguente. In questo modo si ha la sicurezza di osservare la scena in modo corretto. Si noti che se, all'avvenuto collegamento del dispositivo con AirTestIDE, la memoria RAM satura il problema è dovuto al modo in cui vengono allocate le risorse per emulare la grafica del dispositivo. Per risolvere tale problema è necessario cambiare dispositivo emulato come mostrato in [24] nella sezione "Test".

Figura 7.12: Orientamento corretto del marker

Per poter accedere alla gerarchia di oggetti dell'applicazione è necessario selezionare "Unity" nella finestra Poco di AirTestIDE. Ciò ha permesso di capire come gli oggetti fossero stati organizzati dallo sviluppatore e su quali di essi era necessario concentrarsi per effettuare i tests.



a) Gerarchia applicazione marker non riconosciuto

b) Gerarchia applicazione marker riconosciuto

Figura 7.13: Gerarchia oggetti prima e dopo il riconoscimento del marker

Si è subito notato che gli elementi riguardanti la GUI (bottoni e pad), anche se non visibili, causa non inquadramento del marker, erano comunque istanziati e settati come visibili. Ciò ha reso più complicato del previsto il test relativi alla comparsa della scena. Si fa inoltre presente che lo spostamento del dispositivo mobile deve necessariamente essere effettuato a mano. Di conseguenza i test relativi al disconoscimento del marker non potranno essere effettuati.

Script test suite All-States Coverage

Nelle figure 7.14 e 7.15 si mostra lo script scaturito dalla test suite presentata nella sezione 7.1.3. Si fa presente che per effettuare questa tipologia di test è sufficiente scegliere solo uno dei quattro paths precedentemente determinati. La scelta è ricaduta su quello che effettua il movimento verso sinistra e il path seguito è quello ottenuto dall'ordi-

ne in cui sono stati inseriti i test case all'interno della relativa tabella (7.5).

```
4  from airtest.core.api import *
5  auto_setup(__file__)
6  from random import *
7  from poco.drivers.unity3d import UnityPoco
8
9
10 def avviaApp():
11     os.system('cmd /c "cd '+os.environ['USERPROFILE']+'\\Desktop\\AirtestIDE\\airtest\\core\\android\\static\\adb\\windows & adb
   shell am start com.abdu.SafariAR/com.unity3d.player.UnityPlayerActivity"')
12     time.sleep(18);
13     inizializza();
14
15
16 def chiudiApp():
17     keyevent("KEYCODE_APP_SWITCH")
18     os.system('cmd /c "cd '+os.environ['USERPROFILE']+'\\Desktop\\AirtestIDE\\airtest\\core\\android\\static\\adb\\windows & adb
   shell am force-stop com.abdu.SafariAR"')
19     os.system('cmd /c "cd '+os.environ['USERPROFILE']+'\\Desktop\\AirtestIDE\\airtest\\core\\android\\static\\adb\\windows & adb
   shell am kill com.abdu.SafariAR"')
20     time.sleep(5)
21
22 def inizializza():
23     global poco;
24     global nomeAnimaleAttualmentePresente;
25
26     poco = UnityPoco();
27     animaleDefault = "Safari_ZebraSkin";
28     nomeAnimaleAttualmentePresente = animaleDefault;
29
30 def verificaComparsaScena():
31     global listaBottoni
32     global pad
33     global listaAnimali
34
35     scenaSavana = poco("terrain").children();
36     comparsaScena = False;
37     listaBottoni = poco(type='Button');
38     pad = poco("MobileJoystick");
39
40     if(len(scenaSavana) > 0 and len(listaBottoni) > 0 and pad.exists()):
41         comparsaScena = True;
42     assert_equal(comparsaScena,True,"scena comparsa dopo rilevazione marker");
43
44     listaAnimali = getListaAnimali(listaBottoni);
45
46 def getListaAnimali(listaBottoni):
47     listaAnimali = [];
48     for bottone in listaBottoni:
49         tipoAnimale = bottone.attr('name');
50         nomeAnimale = "Safari_"+tipoAnimale+"Skin";
51         listaAnimali.append(nomeAnimale);
52     return listaAnimali;
53
54 def verificaAnimaleUnico():
55     global listaAnimali
56     global nomeAnimaleAttualmentePresente
57
58     for nome in listaAnimali:
59         if (nome != nomeAnimaleAttualmentePresente ):
60             specie = nome.split("_")[1].split("Skin")[0];
61             assert_equal(poco(nome).exists(),False,specie+" non presente sulla scena");
```

Figura 7.14: Script di test All-States Coverage pt1

```
def verificaMovimento(direzione,nomeAnimaleAttualmentePresente,pad):
    direzioni = ["su","giu","destra","sinistra"];
    print("Transizione spostamento pad "+ direzioni[direzione]);
    f.write("pad spostato: " + direzioni[direzione]+"\n");
    animale = poco(nomeAnimaleAttualmentePresente);
    posizione = animale.get_position();
    spostamento = False;

    #spostamento su
    if(direzione == 0):
        pad.swipe([0.0, -0.03])
        posizioneSpostamento = animale.get_position();
        if(posizione[1]>posizioneSpostamento[1]):
            spostamento = True;
    #spostamento giu
    elif(direzione == 1):
        pad.swipe([0.0, 0.03])
        posizioneSpostamento = animale.get_position();
        if(posizione[1]<posizioneSpostamento[1]):
            spostamento = True;
    #spostamento a destra
    elif(direzione == 2):
        pad.swipe([0.03, 0.0])
        posizioneSpostamento = animale.get_position();
        if(posizione[0]<posizioneSpostamento[0]):
            spostamento = True;
    #spostamento a sinistra
    else:
        pad.swipe([-0.03, 0.0])
        posizioneSpostamento = animale.get_position();
        if(posizione[0]>posizioneSpostamento[0]):
            spostamento = True;

    assert_equal(spostamento,True,"spostatamento avennuto nella direzione:"+direzioni[direzione])



poco = None;
listaBottoni = None;
pad = None;
listaAnimali = None;
nomeAnimaleAttualmentePresente = None;
f = open("AllStateCoverageInterazioni.txt","w+");

f.write("Unico paths All States Coverage \n");
avviaApp();
verificaComparsaScena();
verificaAnimaleUnico();
verificaMovimento(3,nomeAnimaleAttualmentePresente,pad);

f.write("Test concluso senza errori \n");
chiudiApp()
```

Figura 7.15: Script di test All-States Coverage pt2

Lo script si compone di diverse funzioni:

- avviaApp: ha l'onere di avviare l'applicazione utilizzando i comandi bash ADB. Successivamente è atteso un tempo pari a 18 secondi, necessario per l'avvio dell'applicazione ed è richiamata la funzione "inizializza". Tale funzione inizializza l'animale di default, considerandolo come quello attualmente presente sulla

scena, e la variabile "poco" utile per stabilire la connessione rpc tra l'IDE e il dispositivo emulato;

- chiudiApp: ha l'onere di chiudere l'applicazione mediante comandi bash ADB che mettono in pausa l'applicazione e poi chiudono il processo in background. Infine si attendono 5 secondi;

- verificaComparsaScena: ha l'onere di verificare che, una volta inquadrato il marker, la scena di realtà aumentata sia creata. Ciò è realizzato verificando che: sia presente l'ambientazione "Savana", assicurandosi che il GameObject "terrain" presenti le componenti necessarie alla realizzazione dell'ambientazione, sia presente la lista di bottoni e il pad. Se tutte queste condizioni sono soddisfatte l'assert è verificato con successo e si può concludere che l'ambientazione di realtà aumentata è stata creata. In seguito viene creata la lista contenente tutti gli animali mediante la funzione "getListaAnimali". Tale funzione riceve come parametro la lista di bottoni, dalla quale crea una lista di stringhe necessarie per riferirsi alla "pelle" degli animali. La motivazione della creazione di tale lista sarà spiegata nella successiva funzione;

- verificaAnimaleUnico: tale funzione ha l'onere di verificare l'univocità dell'animale presente sulla scena. Ciò è realizzato iterando sulla lista di stringhe contenente la "pelle" degli animali, verificando, mediante assert, che qualsiasi altro animale, diver-

samente da quello non selezionato come attualmente presente sulla scena, debba avere il GameObject relativo alla pelle non istanziato. La scelta di effettuare questa implementazione è dovuta al fatto che ogni animale risultava essere già istanziato, ma compariva sulla scena solo conseguentemente all'istanziazione del GameObject relativo alla sua "pelle". Nel caso in esame, l'unico GameObject ad essere istanziato è quello relativo alla zebra, "Safari_ZebraSkin".

- verificaMovimento: ha l'onere di effettuare il movimento e successivamente di verificarne la correttezza. La funzione presenta in ingresso tre parametri: il primo è un intero, il secondo è l'animale presente sulla scena e il terzo è il pad. L'intero rappresenta uno dei quattro spostamenti del pad. In altre parole ciò che si sta facendo è scegliere la direzione che permette il passaggio dallo stato "Scena con animale" a quello di "Animale mosso". Nel corpo di "verificaMovimento" è stato inserito un vettore di direzioni, le quali sono ordinate nel seguente modo: su, giù, destra, sinistra. Il numero passato come parametro è atto a scegliere la direzione dello spostamento e, successivamente, è effettuato il movimento del pad nella direzione selezionata. Infine si verifica mediante assert se lo spostamento è avvenuto nel modo corretto.

Per verificare la correttezza del movimento è necessario spiegare come sono gestite le coordinate dello schermo in AirTestIDE.

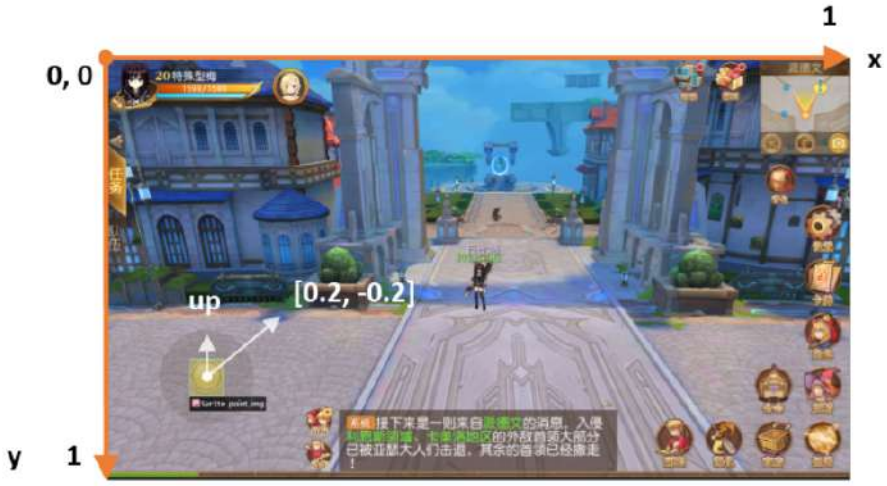


Figura 7.16: Orientamento assi [2]

Come mostrato nella figura 7.16, l'origine degli assi si trova nell'angolo in alto a sinistra dello schermo. Le coordinate sono normalizzate a 1 e riguardo la x sono crescenti verso destra mentre per la y sono crescenti verso il basso. Detto ciò, per verificare la posizione dello spostamento, nella funzione "verificaMovimento" si salvano le coordinate dell'animale prima e dopo lo spostamento, in particolare:

- Per lo spostamento a destra: si valuta se la coordinata x della posizione iniziale risulti inferiore rispetto a quella della posizione finale;

- Per lo spostamento a sinistra: si valuta se la coordinata x della posizione iniziale risulti maggiore rispetto a quella della posizione finale;

- Per lo spostamento giù: si valuta se la coordinata y della posizione iniziale risulti inferiore rispetto a quella della posizione

finale;

- Per lo spostamento su: si valuta se la coordinata y della posizione iniziale risulti maggiore rispetto a quella della posizione finale.

Definite le funzioni necessarie per lo script, si fa presente che "verificaComparsaScena" e "verificaAnimaleUnico" sono gli oracoli per verificare la correttezza dello stato "Scena con animale" mentre la funzione "verificaMovimento" è l'oracolo per stabilire la correttezza dello stato "Animale mosso". Detto ciò, si passa a specificare l'ordine in cui sono state invocate. Innanzi tutto è necessario avviare l'applicazione, avendo predisposto il dispositivo in modo tale che all'avvio di "Safari Animal AR" la videocamera inquadri il marker. A questo punto si verifica se la scena di realtà aumentata sia comparsa e conseguentemente si verifica anche l'unicità dell'animale di default (zebra). Infine si effettua lo spostamento verso sinistra, coerentemente con il path scelto, passando come intero il numero 3 e, in ultimo, si chiude l'applicazione. Tuttavia guardando lo script di test si possono facilmente individuare delle istruzioni di scrittura su file. Esse sono necessarie a chiarire quali interazioni sono state fatte con l'applicazione, cosicché in caso di errore si possa risalire ancora più facilmente alla sequenza di interazioni effettuate. In particolare se il file contenente tali interazioni mostra la presenza, nell'ultima riga, della stringa "Test concluso senza errori" si può dedurre che il test non ha trovato alcun malfunzionamento, e sarà supportato dal report generato da AirTestIDE. Contrariamente se il

file non presenta come ultima stringa quella precedentemente citata, il test presenta errori e sarà supportato dal report di AirTestIDE che definirà quale assert ha fallito. Per ripetere il test sarà necessario o rieseguire tutto lo script oppure eseguire manualmente le istruzioni che hanno portato alla scoperta del malfunzionamento. Nella fattispecie del path implementato dallo script, di seguito si mostrano il report e il file contenente le interazioni effettuate.

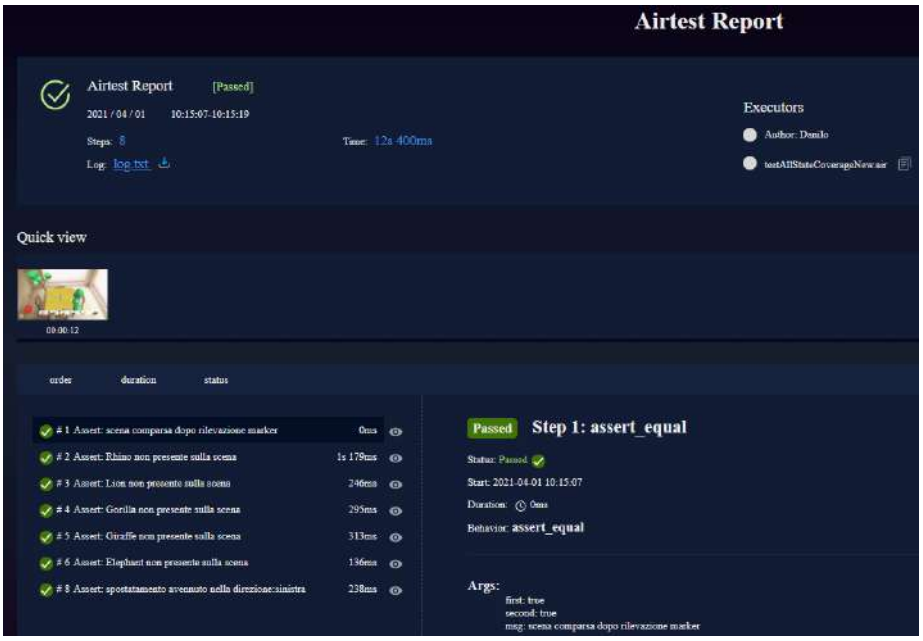


Figura 7.17: Report test suite All-States Coverage

```
Unico paths All States Coverage
pad spostato: sinistra
Test concluso senza errori
```

Figura 7.18: File interazioni All states Coverage

Si noti che si è evitato di riportare la tabella 7.5 poiché la colonna relativa all'esito risulterebbe essere tutta compilata con il valore "passato".

Script test suite All-Transitions Coverage

Come mostrato nel primo script, tutti i casi di test sono andati a buon fine. Tuttavia la precedente tipologia di test non risulta essere esaustiva in quanto si è testata esclusivamente una particolare sequenza di transizioni per verificare la correttezza di tutti gli stati possibili dell'applicazione. Contrariamente l'obiettivo che ci si pone con tale suite di test è il testing di tutte le possibili transizioni con la conseguente verifica degli stati. Nella figura 7.19 si mostra parte dello script scaturito dalla test suite presentata nella tabella 7.8. La motivazione per cui è stato mostrato solo una porzione dello script è da ricondurre al fatto che la restante parte dello script contiene le funzioni già mostrate nella sezione precedente. Quindi la nuova funzione implementata è la seguente:

- clickBottone: ha l'onere di effettuare il click sul bottone verificando che l'animale selezionato compaia sulla scena. La funzione riceve in ingresso il bottone da cliccare, successivamente se l'animale associato a tale bottone non è l'animale attualmente presente, si valuta la sua assenza dalla scena e dopo aver cliccato il bottone si verifica la sua comparsa. Infine si aggiorna l'animale attualmente presente sulla scena.

```
def clickBottone(bottone):

    global animale;
    global nomeAnimaleAttualmentePresente
    global listaAnimali

    tipoAnimale = bottone.attr('name');
    f.write("bottone cliccato: " + tipoAnimale+"\n");
    nomeAnimale = "Safari_"+tipoAnimale+"Skin";
    animale = poco(nomeAnimale);
    if( nomeAnimale != nomeAnimaleAttualmentePresente):
        assert_equal(animale.exists(),False,tipoAnimale+" attualmente non presente sulla scena");
    bottone.click();
    assert_equal(animale.exists(),True,tipoAnimale+" presente sulla scena");
    nomeAnimaleAttualmentePresente = nomeAnimale;


constSu = 0;
constgiu = 1;
constDestra = 2;
constSinistra = 3;

poco = None;
listaBottoni = None;
pad = None;
listaAnimali = None;
nomeAnimaleAttualmentePresente = None;

f = open("AllTransitionsCoverageInterazioni.txt","w+");

f.write("Unico paths All Transitions Coverage \n");

avviaApp();
verificaComparsaScena();
verificaAnimaleUnico();

for bottone in listaBottoni:
    clickBottone(bottone)
    verificaAnimaleUnico();

for index in range(len(listaBottoni)):
    if (index > 3):
        verificaMovimento(constSu,nomeAnimaleAttualmentePresente,pad);
    else:
        verificaMovimento(index,nomeAnimaleAttualmentePresente,pad)

    if (index == 0):
        verificaMovimento(constSu,nomeAnimaleAttualmentePresente,pad);
        verificaMovimento(constgiu,nomeAnimaleAttualmentePresente,pad);
        verificaMovimento(constDestra,nomeAnimaleAttualmentePresente,pad);
        verificaMovimento(constSinistra,nomeAnimaleAttualmentePresente,pad);

    clickBottone(listaBottoni[index])
    verificaAnimaleUnico();

f.write("Test concluso senza errori \n");
chiudiApp();
```

Figura 7.19: Porzione script test suite All-Transitions Coverage

Per ottimizzare il tempo di esecuzione dei test, si è deciso di non eseguirli nell'ordine in cui sono stati inseriti nella tabella, ma di sfruttare lo stato in cui vi si trova per coprire quante più transizioni possibili. Il path seguito è il seguente:

1. Dallo stato "Attesa" si passa allo stato "Scena con animale" mediante l'unica transizione possibile di riconoscimento del marker;

2. Essendo nello stato "Scena con animale" si percorre il cappio parametrizzato per ognuno dei valori possibili;

3. Dallo stato "Scena con animale" si passa ad "Animale mosso" mediante la transizione parametrizzata di movimento del pad scegliendo come direzione dello spostamento "su";

4. Essendo nello stato "Animale mosso" si percorre il cappio parametrizzato per ognuno dei valori possibili;

5. Infine si crea un loop tra lo stato "Animale mosso" e "Scena con animale", andando: dal primo verso il secondo mediante la scelta di uno dei possibili valori della transizione parametrizza non scelti in precedenza; dal secondo verso il primo mediante uno dei possibili valori della transizione parametrizzata non scelti in precedenza. Dato che i possibili valori della transizione parametrizzata che porta da "Animale mosso" a "Scena con animale" sono sei, il loop sarà percorso sei volte. Tuttavia siccome i possibili valori della transizione parametrizzata che porta da "Scena con animale" ad "Animale mosso" sono 4, di cui uno già utilizzato al punto 2, non appena saranno stati tutti percorsi si sceglierà la transizione della direzione su.

Definita l'idea generale si passa a dettagliare il codice in grado di implementare quanto detto in precedenza. Per prima cosa si avvia l'applicazione, verificando la comparsa della scena AR e l'univocità dell'animale di default. Successivamente si itera sulla lista di bottoni richiamando, per ognuno di essi, la funzione "clickBottone" e verificando l'univocità dell'animale presente. Dopodiché si itera nuovamente sulla lista di bottoni per la creazione del loop descritto al punto 5. Infine si chiude l'applicazione. Ugualmente al caso precedente, anche qui sono presenti le istruzioni di scrittura su file utili per definire la sequenza di operazioni effettuate. Di seguito si mostra il report e il file di iterazioni che testimoniano la conclusione del test senza errori. Si noti che, anche in questo caso, si è evitato di riportare la tabella 7.8 poiché la colonna relativa all'esito risulterebbe essere tutta compilata con il valore "passato". L'unica eccezione sono i test relativi al disconoscimento del marker che, come anticipato, non si è in grado di eseguirli mediante script.

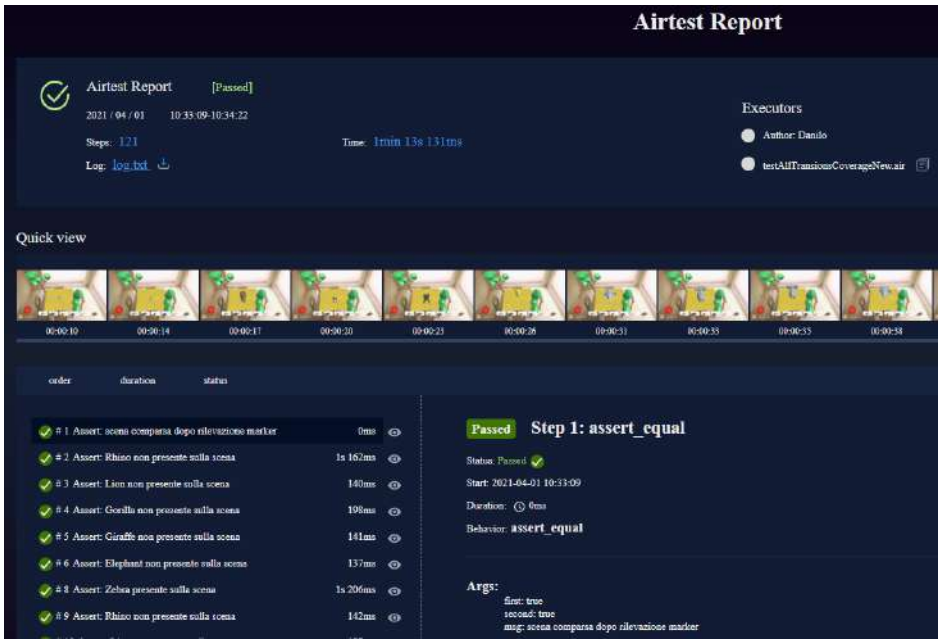


Figura 7.20: Report test suite All-Transitions Coverage

```
Unico paths All Transitions Coverage
bottone cliccato: Zebra
bottone cliccato: Rhino
bottone cliccato: Lion
bottone cliccato: Gorilla
bottone cliccato: Giraffe
bottone cliccato: Elephant
pad spostato: su
pad spostato: su
pad spostato: giu
pad spostato: destra
pad spostato: sinistra
bottone cliccato: Zebra
pad spostato: giu
bottone cliccato: Rhino
pad spostato: destra
bottone cliccato: Lion
pad spostato: sinistra
bottone cliccato: Gorilla
pad spostato: su
bottone cliccato: Giraffe
pad spostato: su
bottone cliccato: Elephant
Test concluso senza errori
```

Figura 7.21: File interazioni All-Transitions Coverage

Script test suite All-One-Loop-Paths Coverage

Nelle figure 7.22 e 7.23 è mostrata la porzione di script relativa alla test suite presentata nella sezione 7.1.3. Diversamente dalle test sui-

te precedenti che hanno testato la correttezza degli stati o di tutte le transizioni (di fatto includendo anche il testing degli stati), in questa test suite l'obiettivo posto è quello di analizzare tutti i possibili paths con una ripetizione di stato. Nella sezione 7.1.3 si era ipotizzato, nella fattispecie di "Safari Animal AR", che tale test suite fosse un caso ridondante della test suite scaturita dal criterio All-Transitions Coverage. Di conseguenza, per dimostrare quanto detto, è stato necessario implementarla al fine di poterla eseguire in modo tale da ottenere dati confrontabili con quelli relativi alla test suite All-Transitions Coverage.

```
poco = None;
listaBottoni = None;
pad = None;
listaAnimali = None;
nomeAnimaleAttualmentePresente = None;

f = open("AllOneLoopPathsCoverageInterazioni.txt","w+");

f.write("PATHS LUNGHEZZA 3 \n");
f.write("\npath 1\n");
avviaApp();
#animale = poco(animaleDefault);
verificaComparsaScena();
verificaAnimaleUnico();


#PATHS DI LUNGHEZZA TRE 6
for ind in range(len(listaBottoni)):
    clickBottone(listaBottoni[ind]);
    verificaAnimaleUnico();
    chiudiApp();
    avviaApp();
    verificaComparsaScena();
    verificaAnimaleUnico();
    if(ind < len(listaBottoni)-1):
        f.write("\npath "+str(ind+2)+"\n");

f.write("\nPATHS LUNGHEZZA 4 \n");
f.write("path 1\n");
count = 2;

```

Figura 7.22: Script test suite All-One-Loop-Paths Coverage pt.1

```
151  # PATHS DI LUNGHEZZA QUATTRO 24 + 16 +24
152  for ind in range(len(listaBottoni)):
153      for direction in range(4):
154
155          clickBottone(listaBottoni[ind]);
156          verificaAnimaleUnico();
157          verificaMovimento(direction,nomeAnimaleAttualmentePresente,pad);
158          chiudiApp();
159          avviaApp();
160          verificaComparsaScena();
161          verificaAnimaleUnico();
162
163          f.write("\npath "+str(count)+"\n");
164          count = count+1
165
166  for i in range(4):
167      for j in range(4):
168          verificaMovimento(i,nomeAnimaleAttualmentePresente,pad);
169          verificaMovimento(j,nomeAnimaleAttualmentePresente,pad);
170          chiudiApp();
171          avviaApp();
172          verificaComparsaScena();
173          verificaAnimaleUnico();
174          f.write("\npath "+str(count)+"\n");
175          count = count+1
176
177
178  for i in range(4):
179      for j in range(len(listaBottoni)):
180          verificaMovimento(i,nomeAnimaleAttualmentePresente,pad);
181          clickBottone(listaBottoni[j]);
182          verificaAnimaleUnico();
183          chiudiApp();
184          if (i != 3 or j != len(listaBottoni)-1):
185              avviaApp();
186              verificaComparsaScena();
187              verificaAnimaleUnico();
188              f.write("\npath "+str(count)+"\n");
189              count = count+1
190
191
192  f.write("Test concluso senza errori \n");
```

Figura 7.23: Script test suite All-One-Loop-Paths Coverage pt.2

La porzione di script mostrata nelle figure 7.22 e 7.23, mostra solo l'ordine con cui sono state richiamate le funzioni. Si è evitato di riportare lo screen ad esse relativo poiché sono esattamente le stesse di quelle precedentemente mostrate. Diversamente dalle test suite precedenti, in questa è prevista l'esecuzione di diversi paths. Ogni paths necessita che l'applicazione sia appena avviata, al fine di partire dalla situazione iniziale senza alcun tipo di alterazione. Riguardo l'ordine di esecuzione dei paths, si è scelto di eseguire prima quelli di lun-

ghezza tre (6) inerenti alla ripetizione dello stato "Scena con animale" mediante il cappio parametrizzato in tale stato. Essi sono relativi al primo ciclo iterativo sulla lista di bottoni, in cui, dopo aver avviato l'applicazione e verificato la corretta comparsa della scena AR e dell'unicità dell'animale, ad ogni iterazione si clicca un bottone non cliccato in precedenza e si verifica l'unicità della presenza dell'animale selezionato sulla scena. In seguito si chiude l'applicazione e la si riapre, verificando nuovamente la comparsa della scena e l'unicità dell'animale di default. Successivamente si è passati ai paths di lunghezza quattro:

- alcuni dei quali (24) ottenuti a partire dai precedenti a cui si è aggiunto lo stato "Animale mosso" mediante il movimento del pad. Essi sono relativi al secondo ciclo iterativo in cui è presente un altro ciclo innestato. Quello più esterno è relativo ai bottoni mentre quello interno alle quattro possibili direzioni. Per ognuna delle iterazioni si clicca il bottone relativo al ciclo iterativo più esterno, si verifica l'unicità dell'animale selezionato, si effettua il movimento secondo l'indice del ciclo iterativo più interno (verificandone la correttezza), si chiude l'applicazione e la si avvia nuovamente verificando la comparsa della scena AR e l'unicità dell'animale di default;

- altri (16) relativi alla ripetizione dello stato "Animale mosso" mediante il cappio parametrizzato in tale stato. Essi sono relativi al terzo ciclo iterativo che contiene un altro ciclo iterativo

innestato. Quello più esterno è relativo alle possibili transizioni dallo stato "Scena con animale" verso "Animale mosso", quello più interno è relativo ai cappi dello stato "Animale mosso". Per ognuna delle iterazioni si effettua il movimento relativo al ciclo iterativo più esterno (verificandone la correttezza), si effettua un nuovo movimento (verificandone la correttezza) questa volta relativo al ciclo iterativo più interno, si chiude l'applicazione e la si riapre verificando la comparsa della scena AR e l'unicità dell'animale di default;

- infine gli ultimi (24) relativi alla ripetizione dello stato "Scena con animale" mediante la transizione parametrizzata di click del bottone dallo stato "Animale mosso". Essi sono relativi al quarto ciclo iterativo che contiene un altro ciclo iterativo innestato. Quello più esterno è relativo alle possibili transizioni dallo stato "Scena con animale" verso "Animale mosso", quello più interno è relativo alle transizioni dei bottoni dallo stato "Animale mosso" a "Scena con animale". Per ognuna delle iterazioni si effettua il movimento relativo al ciclo iterativo più esterno (verificandone la correttezza), si effettua il click del bottone questa volta relativo al ciclo iterativo più interno, si verifica l'unicità dell'animale selezionato, si chiude l'applicazione e, se non si tratta dell'ultima iterazione, la si riapre verificando la comparsa della scena AR e l'unicità dell'animale di default.

Anche qui, come nelle test suite precedenti, sono presenti le istruzioni di scrittura su file per sapere quali interazioni sono state effettuate con l'applicazione. Diversamente dai casi precedenti, qui si eseguono più paths e per questa ragione nel file saranno presenti le interazioni relative ad ognuno di essi. Di conseguenza, in caso di errore, le uniche interazioni che dovrebbero essere effettuate risulterebbero quelle relative all'ultimo path. Altrimenti, nel caso in cui non vi fosse nessun errore, il file presenterebbe come ultima riga la stringa "Test concluso senza errori". In totale sono stati eseguiti 70 paths rispetto ai 75 teorizzati nella sezione 7.1.3. I restanti 5 contengono la transizione di disconoscimento del marker e, come anticipato, non si è in grado di effettuare in modo automatico. Di seguito si mostra il report e il file delle interazioni (parte iniziale e finale, causa eccessiva lunghezza) della test suite appena descritta.

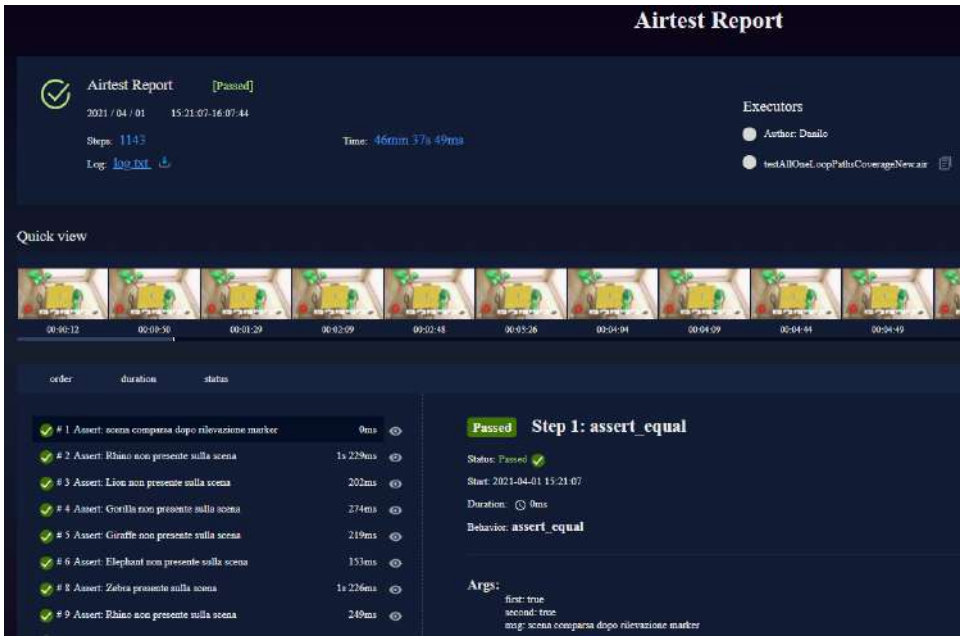


Figura 7.24: Report test suite All-One-Loop-Paths Coverage

Figura 7.25: File interazioni All-One-Loop-Paths Coverage

7.1.5 Valutazione copertura delle test suite

Eseguiti gli script di test si valuta quale sia l'effettiva copertura dei rami del codice di ciascuna delle test suite. Per fare ciò si utilizza lo strumento presentato nella sezione 6.3.2.

Valutazione copertura test suite All-States Coverage

Per primo si valuta la copertura dei rami relativi alla test suite scaturita del criterio All-States Coverage. Di seguito si mostra l'analisi effettuata sul log importato dal dispositivo mobile emulato.

Figura 7.26: Copertura test suite All-States Coverage

Dato che dall'immagine sono visibili solo alcuni dei rami presenti nel log, si riporta la tabella 7.13 che li mostra tutti. Si farà riferimento al numero di esecuzioni di ogni ramo, che non è da confondere con il numero di test effettuati. Lo scopo di riferirsi al numero di esecuzioni di ogni ramo è da attribuire al fatto che la funzione "Update" è eseguita molte più volte rispetto alle altre, poiché ricerca costantemente le modifiche apportate dall'utente. In altre parole anche se si avviasse soltanto l'applicazione e non si interagisse con essa, la funzione "Update" sarebbe comunque eseguita più e più volte.

Rami presenti nel log
AnimalSpawn.start
AnimalSpawn.start.foreach(Transform t in transform)
AnimalsController.start
AnimalsController.update
AnimalsController.update.if (x != 0 || y != 0)
AnimalsController.update.if (x != 0 || y != 0).else
AutoFocus.OnPaused
AutoFocus.OnVuforiaStarted
AutoFocus.start

Tabella 7.13: Rami eseguiti log All-States Coverage

Come si può vedere dalla figura 7.26, il numero più alto di esecuzioni riguarda la funzione di "Update" dello script "AnimalsController" e dei rami in esso presenti. Nella tabella successiva si mostrano i rami non presenti all'interno del log.

Rami assenti nel log
AnimalsController.update.if (x != 0 && y != 0)
AnimalSpawn.Select
AnimalSpawn.Select.if (index == SelectionIndex)
AnimalSpawn.Select.if (index < 0 || index >= models.Count)
AutoFocus.OnPaused.if (!paused)

Tabella 7.14: Rami non eseguiti log All-States Coverage

Da questa classificazione dei rami è possibile scaturire la percentuale di copertura di ogni script, presentata anche in figura 7.26. In tabella 7.15 è mostrata la percentuale di ogni script e il relativo rapporto da cui scaturisce. Per rapporto s'intende il numero di rami coperti rispetto al numero di rami totali, riferito a ciascuno degli script.

	AutoFocus	**AnimalSpawn**	**AnimalsController**
Test suite All-States Coverage	75% ($\frac{3}{4}$)	40% ($\frac{2}{5}$)	80% ($\frac{4}{5}$)

Tabella 7.15: Percentuale copertura All-States Coverage

I rami della tabella 7.14 di seguito saranno analizzati nel dettaglio per definire a quali script appartengono e le motivazioni per cui non sono stati eseguiti. Per quanto riguarda la percentuale di copertura dei rami del codice quella più bassa è relativa allo script "AnimalSpawn", atto alla gestione del click dei bottoni. Di seguito si mostrano i rami non coperti.

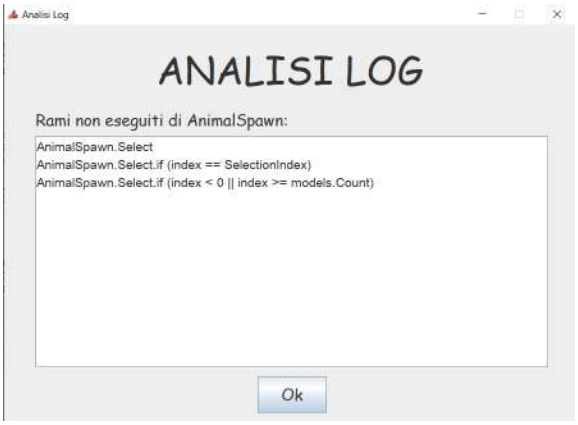


Figura 7.27: Rami non coperti AnimalSpawn All-States Coverage

Come anticipato il metodo "Select", con tutti i relativi rami, non è stato mai eseguito coerentemente con il fatto che, nella test suite, non vi è alcun test riguardante l'interazione di click dei bottoni. Per quanto riguarda i restanti due script, "AnimalsController" e "Autofocus", hanno una copertura abbastanza alta di seguito sono mostrate le immagini dei rispettivi rami non coperti.

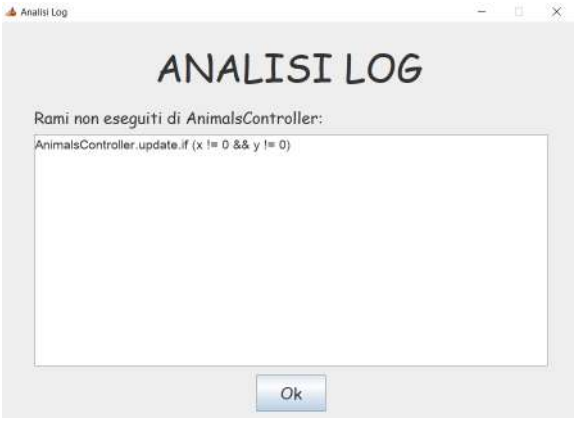


Figura 7.28: Rami non coperti AnimalsController All-States Coverage

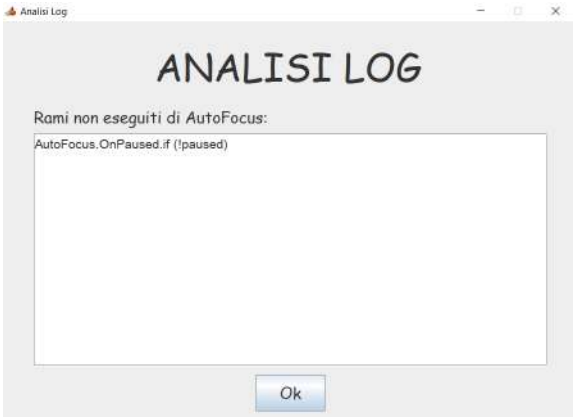


Figura 7.29: Rami non coperti AutoFocus All-States Coverage

Il ramo non coperto di "AnimalsController" è relativo al caso in cui il movimento non parta dalla posizione di default in cui compare l'animale. Per quanto riguarda "AutoFocus" il ramo non coperto è

relativo al caso in cui l'applicazione sia ripresa dopo essere stata messa in pausa.

Valutazione copertura test suite All-Transitions Coverage

Ultimata l'analisi della test suite relativa ad All-States Coverage, si passa ora a quella scaturita dal criterio All-Transitions Coverage. Di seguito si riporta l'analisi ad essa relativa.

Figura 7.30: Copertura test suite All-Transitions Coverage

Dato che dall'immagine sono visibili solo alcuni dei rami presenti nel log, si riporta la tabella 7.16 che li mostra tutti.

Rami presenti nel log
AnimalSpawn.Select
AnimalSpawn.Select.if (index == SelectionIndex)
AnimalSpawn.start
AnimalSpawn.start.foreach(Transform t in transform)
AnimalsController.start
AnimalsController.update
AnimalsController.update.if (x != 0 && y != 0)
AnimalsController.update.if (x != 0 || y != 0)
AnimalsController.update.if (x != 0 || y != 0).else
AutoFocus.OnPaused
AutoFocus.OnVuforiaStarted
AutoFocus.start

Tabella 7.16: Rami eseguiti log All-Transitions Coverage

Come intuibile dalla tabella 7.16, la copertura sarà migliore poiché presenta più rami coperti rispetto alla tabella 7.13. Anche qui, come nel caso precedente, il numero più alto di esecuzioni riguarda la funzione di "Update" dello script "AnimalsController" e dei rami in esso presenti. Il numero maggiore di esecuzioni, rispetto al caso precedente, è dovuto maggior numero di test case (come visibile nella tabella 7.7). Nella tabella successiva si mostrano i rami non presenti all'interno del log.

Rami assenti nel log
AnimalSpawn.Select.if (index < 0 || index >= models.Count)
AutoFocus.OnPaused.if (!paused)

Tabella 7.17: Rami non eseguiti log All-Transitions Coverage

Da questa classificazione dei rami è possibile scaturire la percentuale di copertura di ogni script, presentata anche in figura 7.30. In tabella 7.18 è mostrata la percentuale di ogni script e il relativo rapporto da cui scaturisce. Per rapporto s'intende il numero di rami coperti rispetto al numero di rami totali, riferito a ciascuno degli script.

	AutoFocus	**AnimalSpawn**	**AnimalsController**
Test suite All-States Coverage	75% ($\frac{3}{4}$)	80% ($\frac{4}{5}$)	100% ($\frac{5}{5}$)

Tabella 7.18: Percentuale copertura All-Transitions Coverage

I rami della tabella 7.17 di seguito saranno analizzati nel dettaglio per definire a quali script appartengono e le motivazioni per cui non sono stati eseguiti. Per quanto riguarda la copertura dei rami del codice risulta essere migliore delle precedente poiché: AnimalsController" presenta una copertura totale di tutti i rami in esso presenti; "AnimalSpawn" presenta un incremento significativo rispetto alla precedente test suite, dato che in quella corrente è previsto il click dei bottoni. Come mostrato in figura 7.31, vi è un unico ramo non coperto che risulta essere un controllo nel caso in cui il click di uno dei bottoni fornisse

un indice: o superiore al numero massimo di animali presenti nella lista o inferiore a zero. Tale ramo è un esempio di "dead code", ossia istruzioni che non saranno mai eseguite. Per quanto riguarda "Auto-Focus" mantiene la stessa percentuale e lo stesso ramo non coperto della precedente test suite.

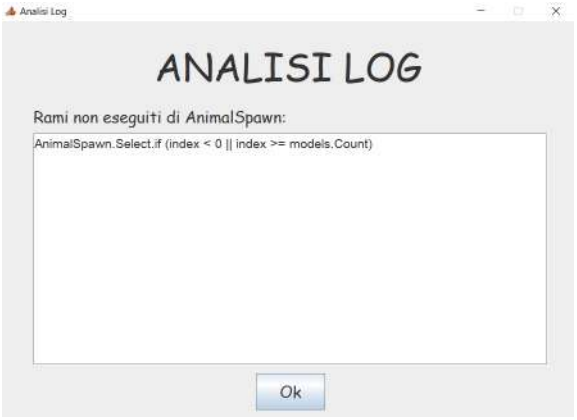


Figura 7.31: Rami non coperti AnimalSpawn All-Transitions Coverage

Valutazione copertura test suite All-One-Loop-Paths Coverage

In ultimo si presenta l'analisi della test suite scaturita dal criterio All-One-Loop-Paths Coverage. Di seguito si riporta la relativa analisi.

Figura 7.32: Copertura test suite All-One-Loop-Paths Coverage

Dato che dall'immagine sono presenti solo alcuni dei rami presenti nel log, in tabella 7.19 se ne mostra l'intera lista.

Rami presenti nel log
AnimalSpawn.Select
AnimalSpawn.Select.if (index == SelectionIndex)
AnimalSpawn.start
AnimalSpawn.start.foreach(Transform t in transform)
AnimalsController.start
AnimalsController.update
AnimalsController.update.if (x != 0 && y != 0)
AnimalsController.update.if (x != 0 || y != 0)
AnimalsController.update.if (x != 0 || y != 0).else
AutoFocus.OnPaused
AutoFocus.OnVuforiaStarted
AutoFocus.start

Tabella 7.19: Rami eseguiti log All-One-Loop-Paths Coverage

Confrontando la tabella 7.19 con la 7.16, si può notare che presentano esattamente gli stessi rami coperti e, di conseguenza, si può intuire che abbiano la stessa copertura (come mostrato in figura 7.32).

Rami assenti nel log
AnimalSpawn.Select.if (index < 0 || index >= models.Count)
AutoFocus.OnPaused.if (!paused)

Tabella 7.20: Rami non eseguiti log All-One-Loop-Paths Coverage

	AutoFocus	**AnimalSpawn**	**AnimalsController**
Test suite			
All-States	75% ($\frac{3}{4}$)	80% ($\frac{4}{5}$)	100% ($\frac{5}{5}$)
Coverage			

Tabella 7.21: Percentuale copertura All-One-Loop-Paths Coverage

Dualmente, come mostrato in tabella 7.20, si avranno anche gli stessi rami non coperti del caso precedente e per questa ragione si evita di spiegarne le motivazioni e di riportarne gli screen. Infine, il ramo con maggior numero di esecuzioni riguarda la funzione di "Update" dello script "AnimalsController" e dei rami in esso presenti. Ovviamente il numero di esecuzioni risulta essere molto maggiore rispetto al caso precedente, dato che il numero di test case di tale test suite è di molto maggiore (7.12).

Confronto tra le test suite

Dai risultati ottenuti nei precedenti paragrafi, si confrontano le test suite per definire quale fra esse risulta essere migliore per il testing dell'applicazione in esame. E' lampante che le test suite scaturite dal criterio All-Transitions Coverage e All-One-Loop-Pathts Coverage risultano essere migliori di quella scaturita dal criterio All-States Coverage, poiché presentano una copertura dei rami di codice nettamente superiore, come mostrato nella tabella 7.22. A questo punto, essendo che la test suite All-Transitions Coverage e All-One-Loop-Pathts Coverage

hanno la stessa copertura, per definire quale tra esse sia la migliore, si valuta l'efficacia attraverso il numero di test case presenti in ognuna delle test suite. Com'è possibile notare dal confronto tra le righe della tabella 7.23, il numero di test case relativo ad All-Transition è decisamente inferiore rispetto a quello associato ad All-One-Loop-Paths e, pertanto, si ritiene la test suite All-Transitions Coverage migliore.

	AutoFocus	**AnimalSpawn**	**AnimalsController**
Test suite All-States Coverage	75% ($\frac{3}{4}$)	40% ($\frac{2}{5}$)	80% ($\frac{4}{5}$)
Test suite All-Transitions Coverage	75% ($\frac{3}{4}$)	80% ($\frac{4}{5}$)	100% ($\frac{5}{5}$)
Test suite All-One-Loop-Paths Coverage	75% ($\frac{3}{4}$)	80% ($\frac{4}{5}$)	100% ($\frac{5}{5}$)

Tabella 7.22: Confronto percentuale copertura

	N° test case
Test suite All-States Coverage	2
Test suite All-Transitions Coverage	23
Test suite All-One-Loop-Paths Coverage	218

Tabella 7.23: Confronto numero test case

Definita tale test suite come la migliore tra quelle proposte, è lecito domandarsi se la copertura di tale test suite possa essere migliorata ancor di più. Per rispondere a questo quesito bisogna analizzare i rami degli script che non risultano avere una copertura totale. Cominciando dallo script "AutoFocus", l'unico ramo che non risulta essere coperto è quello relativo al caso in cui l'applicazione sia messa in pausa e successivamente ripresa. Questa funzionalità è relativa alla categoria di eventi di sistema Android e, come tale, esulava dagli oneri del modello proposto in figura 7.11. Aggiungendo tale test case alla test suite All-Transitions Coverage si raggiunge una copertura completa per lo script in questione. Di seguito si mostrano le righe di codice che implementano tale test case. Esse sono state aggiunte in coda allo script di test di figura 7.19.

```
keyevent("KEYCODE_APP_SWITCH")
keyevent("KEYCODE_APP_SWITCH")
```

Figura 7.33: Codice test case pausa e ripresa applicazione

Effettuando nuovamente l'analisi con lo strumento "Analisi Log", si può vedere che la copertura totale è raggiunta sia per "AnimalsController" che per "AutoFocus", come mostrato in figura 7.34.

Figura 7.34: Nuova copertura All-Transitions Coverage

Diverso è il discorso riguardante il ramo non coperto dello script "AnimalSpawn", infatti tale ramo non potrà essere coperto in nessuna esecuzione dell'applicazione poichè, come anticipato, esso sarà attivato solo nel caso in cui, al click di uno dei bottoni, fosse fornito un indice: o superiore al numero massimo di animali presenti nella lista o inferiore a zero.

	AutoFocus	**AnimalSpawn**	**AnimalsController**
Test suite All-Transitions Coverage estesa	100% $(\frac{4}{4})$	80% $(\frac{4}{5})$	100% $(\frac{5}{5})$

Tabella 7.24: Percentuale copertura test suite All-Transitions estesa

In definitiva la migliore copertura possibile che si è riusciti ad ottenere è quella mostrata nella tabella 7.24.

7.1.6 Analisi Mutazionale

Al fine di valutare la qualità delle test suite ottenute si applicherà l'analisi mutazionale su "Safari Animal AR". Tale analisi sarà effettuata su tutte le test suite e non solo su quella ritenuta migliore, ossia quella scaturita dal criterio All-Transitions Coverage, perché è necessario confrontare la capacità di ognuna di scoprire malfunzionamenti. In altre parole potrebbe essere possibile che l'elevata, o mediocre, copertura di codice non sia correlata alla capacità di scoperta dei malfunzionamenti. Quindi il primo passo necessario per effettuare l'analisi mutazionale consta nella definizione del fault model. Analizzando gli script dell'applicazione unitamente con le action presentate in tabella 7.1, si è pensato di iniettare i seguenti difetti:

1. Invertire direzione dello spostamento rispetto all'asse x;
2. Violazione del vincolo di esclusività di presenza di un animale sulla scena.

Gli operatori assegnati sono rispettivamente: utilizzo dell'operatore aritmetico meno e aggiunta di una nuova istruzione. Come si può intuire, ciò genera due mutanti:

1. Mutante 1: modifica della direzione dello spostamento rispetto all'asse x. Tale modifica impatta sul movimento del pad invertendo le direzioni di spostamento lungo quell'asse. In altre parole, con questa modifica, spostando il pad verso destra l'animale si muoverà a sinistra e viceversa. La modifica è stata effettuata alla riga 41 dello script "AnimalsController", infatti confrontandolo con quello presentato in figura 7.3 si può notare che è stato inserito il simbolo meno, responsabile del suddetto malfunzionamento. Lo script modificato è presentato in figura 7.35;

2. Mutante 2: violazione del vincolo di esclusività di presenza di un animale sulla scena. In altri termini, la zebra è stata resa costantemente presente sulla scena, quindi anche cliccando uno dei bottoni relativo agli altri animali ciò non comporta la sostituzione dell'animale ma l'aggiunta di esso alla zebra. In base a quanto detto sulla scena sarà presente o solo la zebra o la zebra e un altro animale. Questa errore è stato iniettato modificando lo script "AnimalSpawn", inserendo, nella funzione "Select", l'istruzione alla riga 77, come si può notare confrontandolo con quello mostrato in figura 7.2. Lo script modificato è presentato in figura 7.36;

In ultimo è fondamentale presentare gli oracoli al fine di valutare se le test suite in esame sono in grado di uccidere i mutanti. Gli oracoli

che saranno descritti, sono quelli già introdotti nell'implementazione degli script di test, sezione 7.1.4. Di seguito si mostrano gli oracoli per ogni versione dei mutanti:

1. Oracolo 1: è associato al mutante 1 e consiste nel valutare la coerenza della direzione di spostamento, rispetto all'asse x, del pad e dell'animale. In altre parole se il pad è spostato verso destra l'animale si muove verso destra, mentre se è spostato verso sinistra l'animale si muove verso sinistra. A livello pratico ciò sarà verificato, ove presente, mediante l'assert relativo al suddetto movimento, contenuto nella funzione "verificaMovimento";

2. Oracolo 2: è associato al mutante 2 e consiste nel verificare la presenza esclusiva di uno degli animali sulla scena. In altre parole qualsiasi animale, sia esso di default o selezionato mediante uno dei bottoni, deve essere l'unico presente sulla scena visualizzata. A livello pratico ciò sarà verificato, ove presenti, mediante gli assert relativi all'assenza degli altri animali, contenuti nella funzione "verificaAnimaleUnico".

Dopo aver descritto i mutanti, si passa alla valutazione della qualità delle test suite delle precedenti sezioni.

```
18
19      // Define the Animation component
20      private Animation anim;
21
22
23      void Start()
24      {
25
26          // Assign the above variable "rb" to the GameObject's RigidBody component
27          rb = GetComponent<Rigidbody>();
28
29          // Assign the above variable "anim" to the GameObject's Animation component
30          anim = GetComponent<Animation>();
31          SondaManager.inserisciSonda("AnimalsController.start");
32
33      }
34
35      // For each frame
36      void Update()
37      {
38          SondaManager.inserisciSonda("AnimalsController.update");
39          // Get the horizontal axis from the joystick
40          // (CrossPlatformInput is the library which controls the joystick)
41          float x = -CrossPlatformInputManager.GetAxis("Horizontal");
42
43          // Get the vertical axis from the joystick
44          float y = CrossPlatformInputManager.GetAxis("Vertical");
45
46          // Create a new vector, which will hold the position of the joystick
47          Vector3 movement = new Vector3(x, 0, y);
48
49          // Rate of speed the animals will be moving
50          rb.velocity = movement * 0.15f;
51
52          // If joystick's x and y values are different from 0
53          // i.e. joystick has been moved
54          if (x != 0 && y != 0)
55          {
56              // Move the animal in the direction of the joystick
57              transform.eulerAngles = new Vector3(transform.eulerAngles.x, Mathf.Atan2(x, y) * Mathf.Rad2Deg, transform.eulerAngles.z);
58              SondaManager.inserisciSonda("AnimalsController.update.if (x != 0 && y != 0)");
59          }
60
61          // If either x or y values are different from 0
62          // i.e. animal is moving
63          if (x != 0 || y != 0)
64          {
65              // Play the walk animation
66              anim.Play("walk");
67              SondaManager.inserisciSonda("AnimalsController.update.if (x != 0 || y != 0)");
68          }
69
70          // Otherwise
71          // i.e. animal is not being moved
72          else
73          {
74              // Play the idle animation
75              anim.Play("idle");
76              SondaManager.inserisciSonda("AnimalsController.update.if (x != 0 || y != 0).else");
77          }
78      }
79  }
```

Figura 7.35: Script AnimalsController modificato

```
29          models = new List<GameObject>();
30
31          // For each child of the GameObject this script has been applied to
32          // i.e. all the animals under the "SafariAnimalsList" GameObject
33          foreach(Transform t in transform)
34          {
35              SondaManager.inserisciSonda("AnimalSpawn.start.foreach(Transform t in transform)");
36              // Add the animal to the "models" list
37              models.Add(t.gameObject);
38
39              // Set the animal's visibility to hidden
40              // i.e. active = false
41              t.gameObject.SetActive(false);
42          }
43
44          // Only show the animal that has been selected
45          // i.e. upon launching the app, models[0] (which is the zebra) will be the only animal visible
46          models[SelectionIndex].SetActive(true);
47      }
48
49      // Event detecting change of index
50      // i.e. event triggered upon pressing one of the animals from the bottom menu (each entry is a button)
51      public void Select(int index) {
52
53          SondaManager.inserisciSonda("AnimalSpawn.Select");
54
55          // If the selected animal is already displayed on screen
56          if (index == SelectionIndex)
57              {
58                  SondaManager.inserisciSonda("AnimalSpawn.Select.if (index == SelectionIndex)");
59                  // Nothing happens
60                  return;
61              }
62
63
64
65          // If the value of index is different than the number of animals contained in the list
66          // (in case the app is glitched)
67          if (index < 0 || index >= models.Count)
68          {
69              SondaManager.inserisciSonda("AnimalSpawn.Select.if (index < 0 || index >= models.Count)");
70              // Nothing happens
71              return;
72          }
73
74
75          // Otherwise, hide the currently displayed animal
76          models[SelectionIndex].SetActive(false);
77          models[0].SetActive(true);
78          // Change SelectionIndex to the newly selected animal
79          SelectionIndex = index;
80
81          // Display this animal
82          models[SelectionIndex].SetActive(true);
83      }
84  }
```

Figura 7.36: Script AnimalSpawn modificato

Test suite All-States Coverage

Con tale test suite, descritta nella sezione 7.1.3, si valuta quale sia la sua capacità di scoprire i malfunzionamenti, relativi al movimento, indotti nel primo mutante dell'applicazione. Tale test suite è in grado di

individuare il malfunzionamento, dato che la direzione di spostamento scelta è sinistra. Di seguito si mostra il report e il file delle interazioni che confermano quanto detto.

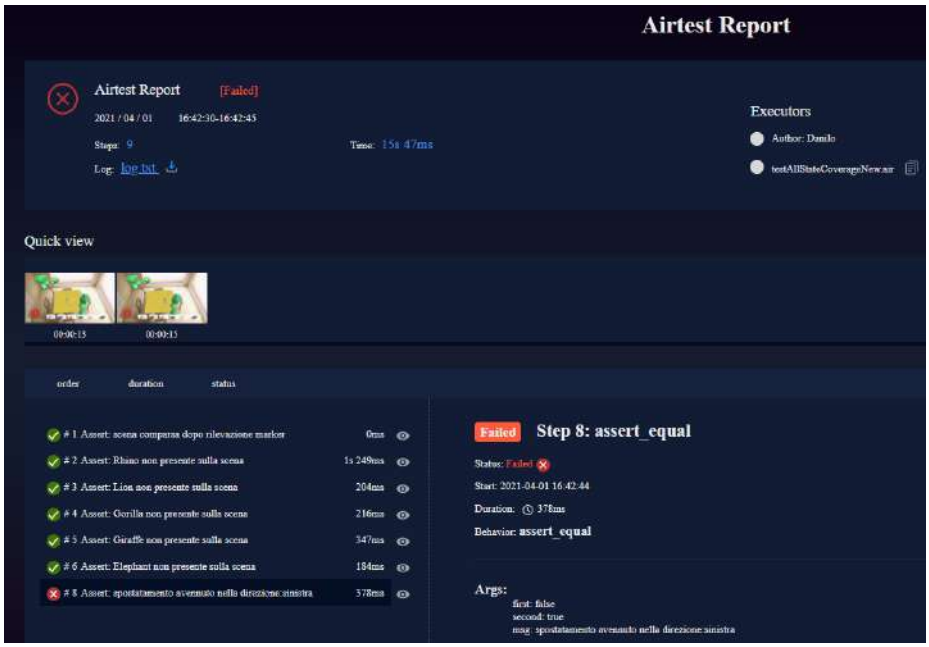


Figura 7.37: Report test suite All-State Coverage con uccisione mutante 1

```
Unico paths All States Coverage
pad spostato: sinistra
```

Figura 7.38: File interazioni All-State Coverage con uccisione mutante 1

Il report mostrato in figura 7.37 segnala correttamente l'errore, infatti l'esito dell'assert relativo allo spostamento non è quello mostrato dall'oracolo di figura 7.39, così come il file delle interazioni 7.38 rispetto a quello dell'oracolo (7.18) non presenta la stringa finale di conclusione con successo del test.

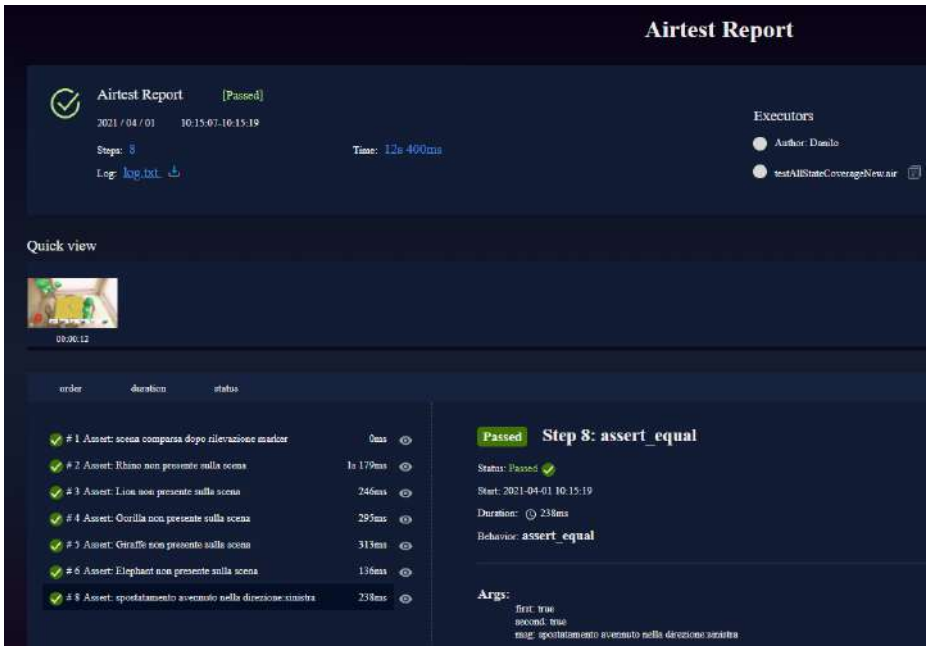


Figura 7.39: Oracolo mutante 1 direzione modificata

Per quanto riguarda il secondo mutante dell'applicazione, che prevede il malfunzionamento di comparsa di più animali sulla scena, tale test suite non sarà mai in grado di rilevarlo dato che non prevede il click dei bottoni e che l'animale di default presente sulla scena è la zebra, unico animale ad essere esclusivamente presente sulla scena. Di seguito si mostra il report del test eseguito sul secondo mutante dell'applicazione. Il file delle interazioni si è evitato di riportarlo poichè sarebbe uguale a quello di figura 7.18.

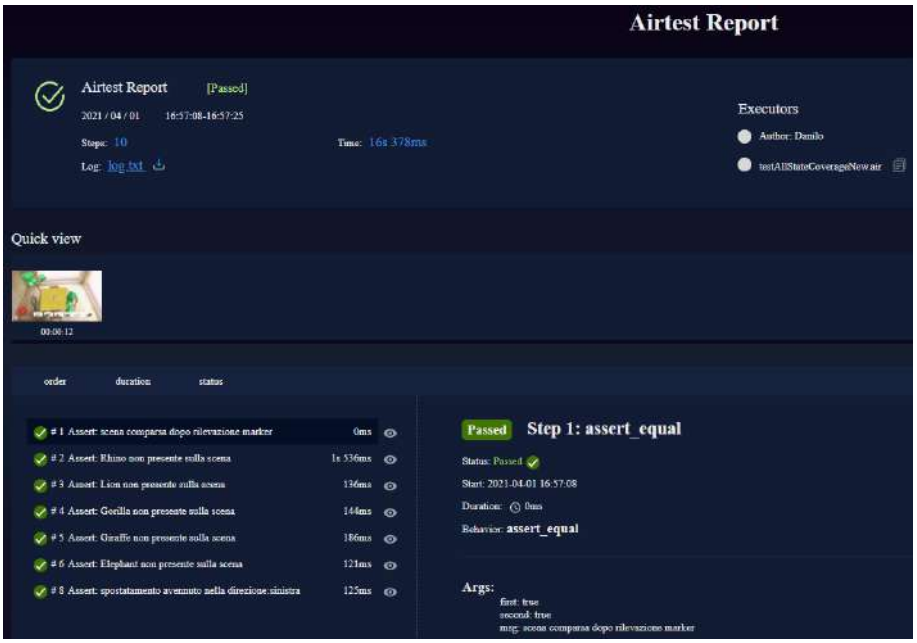


Figura 7.40: Report test suite All-State Coverage senza uccisione mutante 2

Test suite All-Transitions Coverage

Con tale test suite, descritta nella sezione 7.1.3, si valuta la sua capacità di scoprire i malfunzionamenti, relativi al movimento, indotti nel primo mutante dell'applicazione. Tale test suite sarà sicuramente in grado di rilevare l'errore poiché effettuerà tutte le transizioni possibili, e, di conseguenza, indubbiamente effettuerà quelle relative allo spostamento del pad a destra o a sinistra. Di seguito si mostra il report e il file delle interazioni scaturenti dallo script di test applicato a questo mutante dell'applicazione.

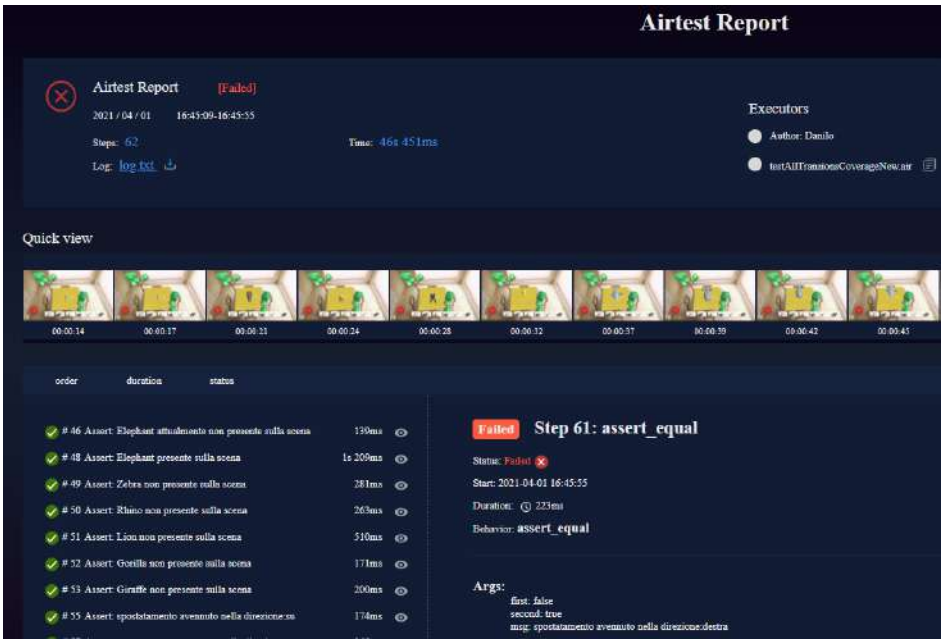


Figura 7.41: Report test suite All-Transition Coverage con uccisione mutante 1

```
Unico paths All Transitions Coverage
bottone cliccato: Zebra
bottone cliccato: Rhino
bottone cliccato: Lion
bottone cliccato: Gorilla
bottone cliccato: Giraffe
bottone cliccato: Elephant
pad spostato: su
pad spostato: su
pad spostato: giu
pad spostato: destra
```

Figura 7.42: File interazioni All transitions Coverage mutante 1

Come si può notare, il report riporta il malfunzionamento rilevato rispetto allo spostamento effettuato. Infatti confrontandolo con l'oracolo di figura 7.43 si può notare che l'assert relativo al suddetto spostamento risulta essere fallito, così come il file delle interazioni 7.42 rispetto a quello dell'oracolo (7.21) non presenta la stringa finale di conclusione con successo del test.

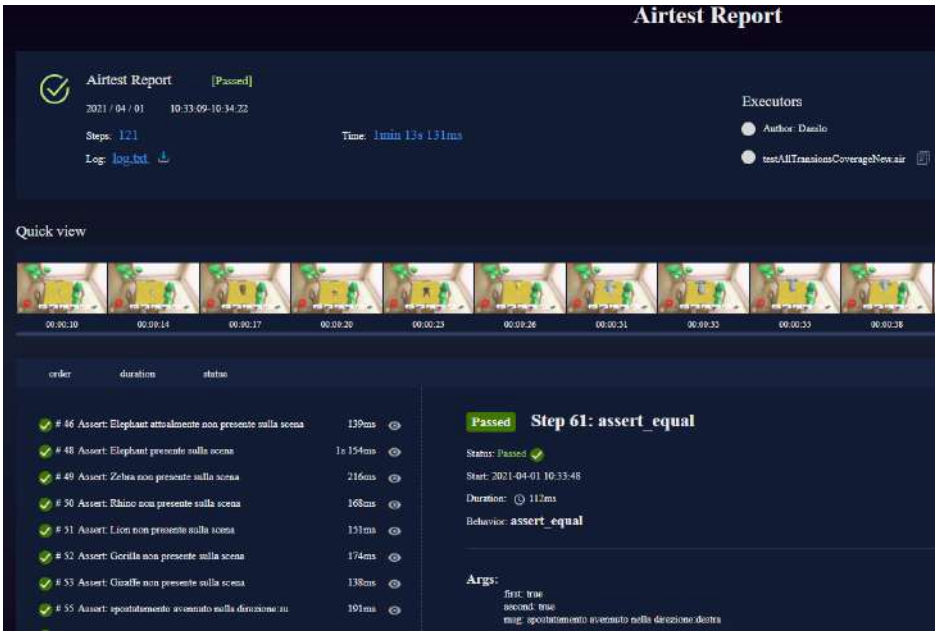


Figura 7.43: Oracolo mutante 1

Per quanto riguarda il secondo mutante di "Safari Animal AR", anche in questo caso la suite di test sarà in grado di riconoscere il malfunzionamento, poiché essa prevede il click dei bottoni con conseguente verifica di comparsa dell'animale scelto e la verifica della sua esclusiva presenza sulla scena. Di seguito si presenta il report e il file delle interazioni che prova quanto detto. Il report di figura 7.44 presenta il fallimento dell'assert di assenza della zebra, contrariamente a quanto avviene nell'assert dell'oracolo di figura 7.46), così come il file delle interazioni 7.45 rispetto a quello dell'oracolo (7.21) non presenta la stringa finale di conclusione con successo del test.

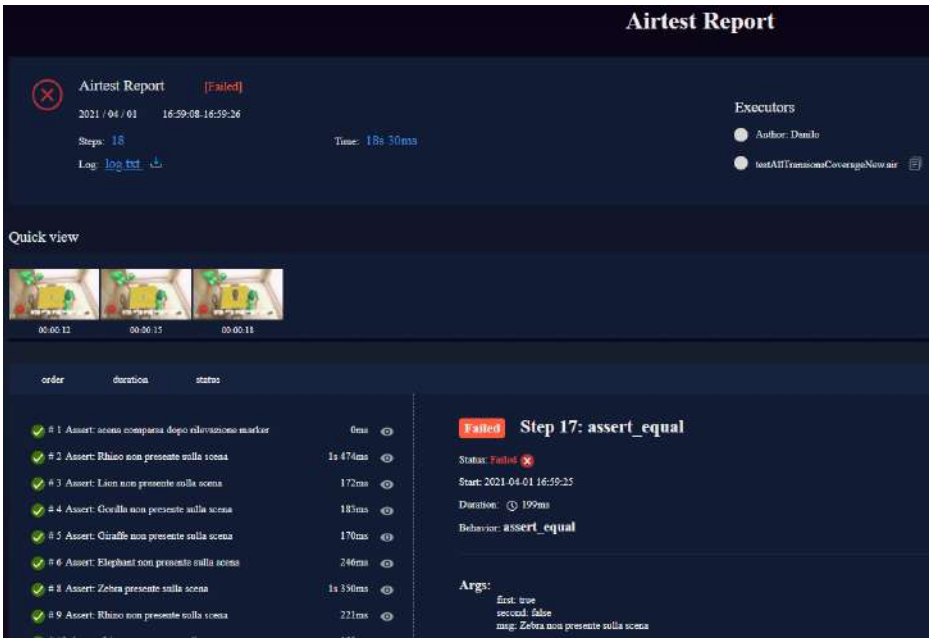


Figura 7.44: Report test suite All-Transition Coverage con uccisione mutante 2

```
Unico paths All Transitions Coverage
bottone cliccato: Zebra
bottone cliccato: Rhino
```

Figura 7.45: File interazioni All Transitions Coverage mutante 2

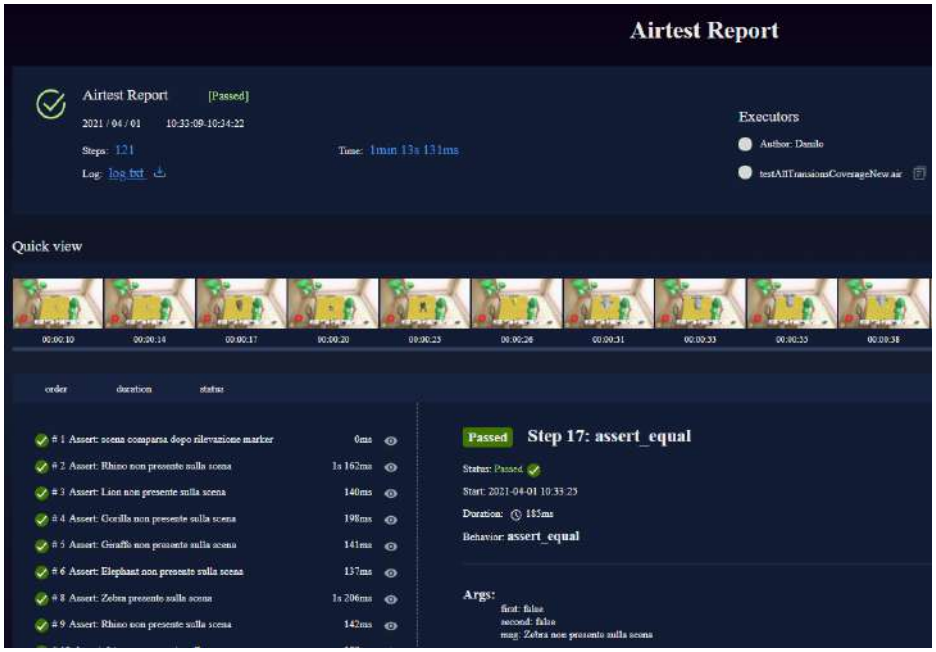


Figura 7.46: Oracolo mutante 2

Test suite All-One-Loop-Pathts Coverage

Con tale test suite, descritta nella sezione 7.1.3, si valuta la sua capacità di scoprire i malfunzionamenti, relativi al movimento, indotti nel primo mutante. Così come la precedente, essa sarà in grado di rilevare l'errore, dato che in diversi paths prevede il movimento nelle direzioni che provocano il malfunzionamento. Di seguito si mostra il report e il file delle interazioni scaturente dallo script di test applicato al suddetto mutante dell'applicazione.

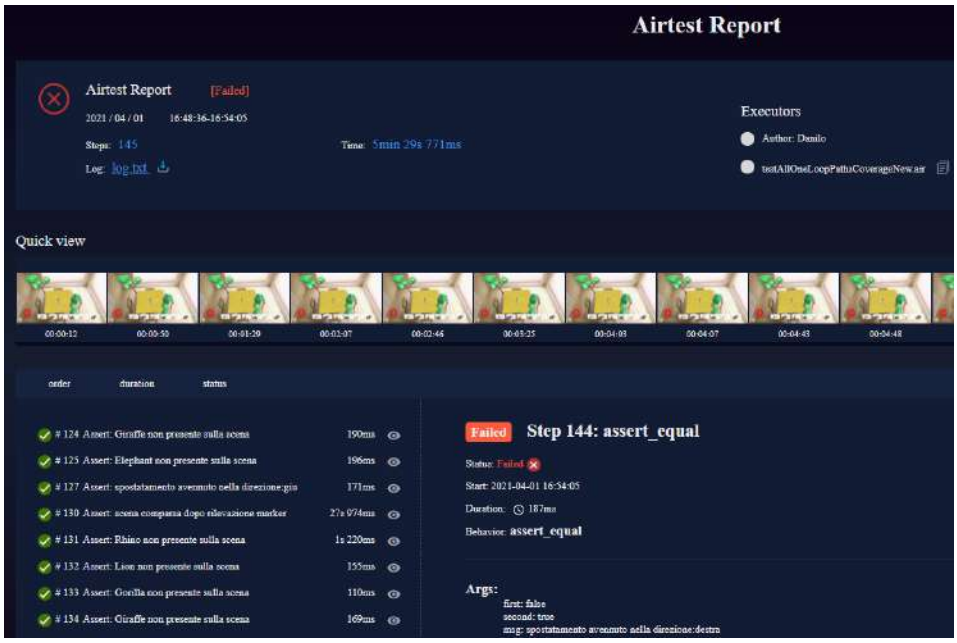


Figura 7.47: Report test suite All-One-Loop-Paths Coverage con uccisione mutante 1

```
PATHS LUNGHEZZA 3

path 1
bottone cliccato: Zebra

path 2
bottone cliccato: Rhino

path 3
bottone cliccato: Lion

path 4
bottone cliccato: Gorilla

path 5
bottone cliccato: Giraffe

path 6
bottone cliccato: Elephant

PATHS LUNGHEZZA 4
path 1
bottone cliccato: Zebra
pad spostato: su

path 2
bottone cliccato: Zebra
pad spostato: giu

path 3
bottone cliccato: Zebra
pad spostato: destra
```

Figura 7.48: File interazioni All-One-Loop-Paths Coverage mutante 1

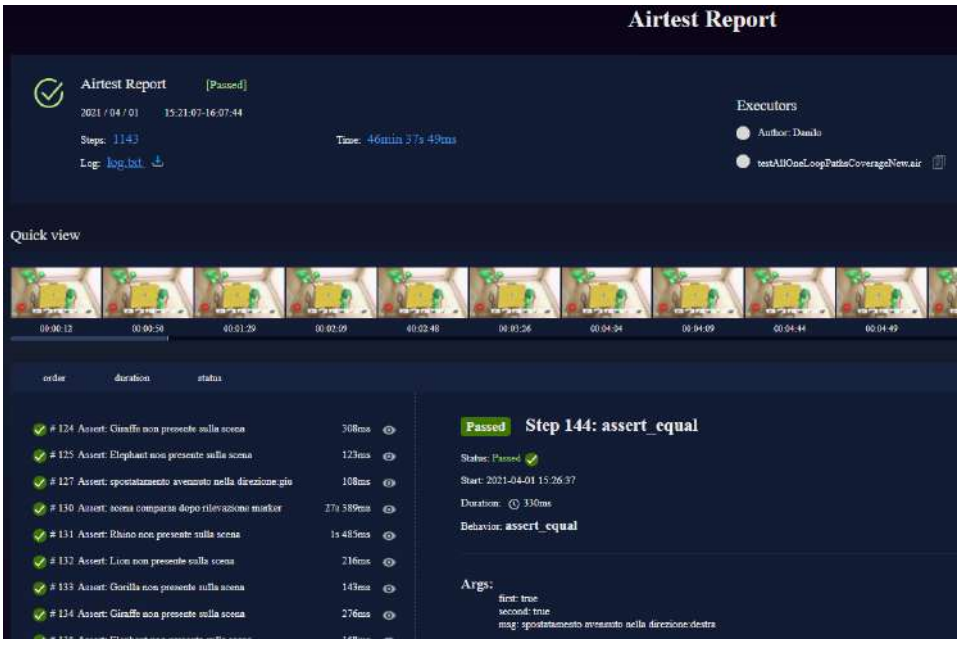


Figura 7.49: Oracolo mutante 1 All-One-Loop-Paths Coverage

Come si può notare, il report di figura 7.47 fallisce a causa dalla

rilevazione del malfunzionamento. Infatti confrontandolo con l'oracolo di figura 7.49, si può notare che l'assert relativo allo spostamento risulta essere fallito, così come il file delle interazioni 7.48 rispetto a quello dell'oracolo (7.25) non presenta la stringa finale di conclusione con successo del test. Passando al secondo mutante dell'applicazione, anche in questo caso la test suite risulterà in grado di riconoscere il malfunzionamento poiché, in diversi paths, prevede il click dei bottoni con la conseguente verifica di comparsa dell'animale scelto e la verifica della sua esclusiva presenza sulla scena. Il report di figura 7.50 presenta il fallimento dell'assert di assenza della zebra, contrariamente a quanto avviene nell'assert dell'oracolo di figura 7.52), così come il file delle interazioni 7.51 rispetto a quello dell'oracolo (7.25) non presenta la stringa finale di conclusione con successo del test.

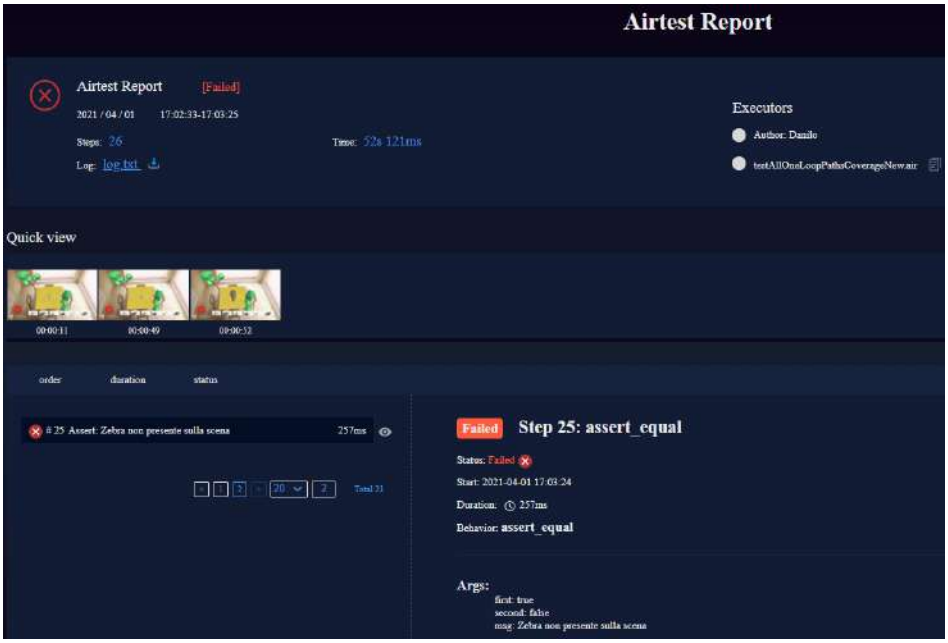


Figura 7.50: Report test suite All-One-Loop-Paths Coverage con uccisione mutante 2

```
PATHS LUNGHEZZA 3

path 1
bottone cliccato: Zebra

path 2
bottone cliccato: Rhino
```

Figura 7.51: File interazioni All-One-Loop-Paths Coverage mutante 2

Figura 7.52: Oracolo mutante 2 All-One-Loop-Paths Coverage

Confronto

Come si è potuto intuire, le test suite scaturite del criterio All-Transitions Coverage e All-One-Loop-Paths Coverage risultano essere migliori di quella scaturita dal criterio All-States Coverage. Tuttavia, per dimostrarlo matematicamente, si effettua il calcolo dell'efficacia di ognuna delle test suite, utilizzando la formula mostrata nella sezione 6.3.3.

Riguardo la test suite di All-States Coverage, il calcolo dell'efficacia è effettuato come segue:

$$TER_{AllStatesCoverage} = \frac{km}{tm} = \frac{(1+0)}{2} = 0.5$$

Passando alla test suite relativa al criterio All-Transitions Coverage si ha che entrambi i mutanti sono uccisi, di conseguenza il calcolo dell'efficacia è effettuato come segue:

$$TER_{AllTransitionsCoverage} = \frac{km}{tm} = \frac{(1+1)}{2} = 1$$

In ultimo si calcola l'efficacia relativa alla test suite All-One-Loop-Paths Coverage, che, come la precedente, è in grado di uccidere entrambi i mutanti. Per tale ragione il calcolo dell'efficacia dà il seguente risultato:

$$TER_{AllOneLoopPathsCoverage} = \frac{km}{tm} = \frac{(1+1)}{2} = 1$$

Come già successo nella precedente sezione anche qui le test suite relative ai criteri All-Transitions Coverage e All-One-Loop-Paths Coverage si equivalgono. Quindi essendo che hanno la stessa efficacia, l'unica differenziazione è possibile effettuarla solo relativamente al numero di test case (tabella di riepilogo 7.23) e, per questa ragione, si ritiene che la migliore test suite, per il caso in esame, sia quella relativa al criterio All-Transitions Coverage.

7.2 Caso di studio 2: PointAR

Il secondo progetto che si propone come caso di studio è "PointAR" che, come il precedente, è scaricabile dal Google Play Store. Il codice sorgente del progetto è stato scaricato da GitHub dal link [25] ed era presente tra i progetti impiegati per l'analisi presentata nel capitolo 5. Anche in questo caso, la scelta del progetto è stata dovuta non solo alla conformità rispetto alle tecnologie target individuate che, come affermato in precedenza, risulta essere una prerogativa fondamentale; ma anche perché vi è una chiara spiegazione del funzionamento e degli script utilizzati. Nonostante ciò non vi è alcuna tipologia di test e per questa ragione si è deciso di testarne il funzionamento. L'applicazione in esame è di tipo marker-based e ha obiettivi pedagogici, in particolare il suo scopo è assistere la forza lavoro straniera nella lettura dei cartelli presenti all'interno del posto di lavoro, offrendo la traduzione in diverse lingue. Una differenza sostanziale rispetto al caso precedente è che "Safari Animal AR" è un'applicazione AR di tipo marker-based in grado di riconoscere esclusivamente un unico marker; invece "PointAR" è un'applicazione marker-based in grado di riconoscere due marker creando per ognuna una scena di realtà aumentata diversa. Il riconoscimento del marker deve avvenire in modo esclusivo, ciò significa che nel caso in cui fossero presenti entrambi i marker, sarebbe creata la scena solo del primo marker riconosciuto. Per quanto riguarda il funzionamento dell'applicazione, una volta avviata è pre-

sente la schermata mostrata in figura 7.53, in cui è possibile selezionare dal menù a tendina una lingua tra: inglese, italiano, lituano e urdu. La lingua di default che compare come prima scelta è l'inglese.

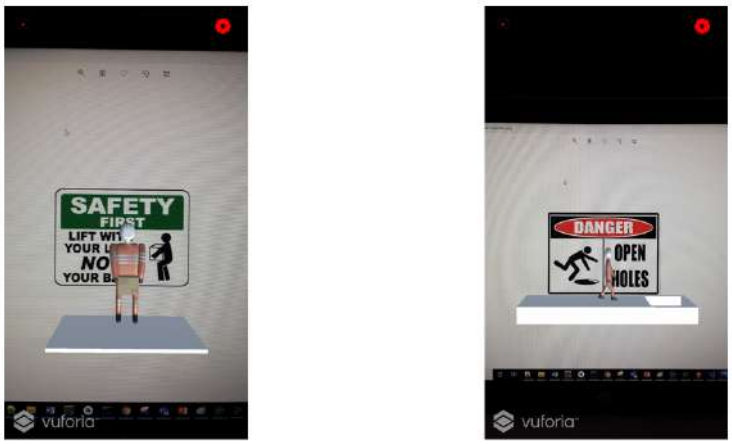
Figura 7.53: Schermata iniziale PointAR

Successivamente cliccando sul tasto "Start", si attiverà la fotocamera e, se uno dei due marker di figura 7.54 sarà inquadrato, saranno mostrate rispettivamente le scene di figura 7.55. Ogni scena è composta dal marker, che sarà tradotto nella lingua selezionata grazie allo script "translate" di figura 7.57 e 7.58, e da un'animazione esplicativa del testo cartello la cui riproduzione è controllata dallo script "playbackspeed" di figura 7.56. Da questo momento in poi, il marker presentato in figura 7.54 corrispondente al caso a) sarà identificato come marker 1, mentre quello corrispondente al caso b sarà identificato come marker 2. Altra interazione che l'utente potrà effettuare è il cambio lingua intanto che la fotocamera è attivata. Ciò potrà effettuarlo cliccando sul bottone dell'ingranaggio, posta in alto a destra, sarà quindi nuo-

vamente possibile selezionare la lingua dal menù a tendina e cliccando sul tasto "Resume" si potrà tornare alla fotocamera. Ovviamente, se la lingua selezionata sarà diversa rispetto a quella precedente, la scena di realtà aumentata varierà la traduzione del cartello mostrato. In ultimo all'interno del progetto è presente lo script "AutoFocus", figura 7.59, che è necessario per il controllo dell'orientamento del dispositivo.

a) Marker 1 b) Marker 2

Figura 7.54: Marker PointAR



a) Scena AR 1 b) Scena AR 2

Figura 7.55: Scene AR PointAR

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class playbackspeed : MonoBehaviour
{
    //Define Animation component
    public Animation anim;

    void Start()
    {
        // Set playback speed of "lift" animation to 0.25 (floating point)
        anim["lift"].speed = 0.25f;
    }
}
```

Figura 7.56: Script playbackspeed

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class translate : MonoBehaviour
{
    // Define the dropdown GameObject
    public Dropdown myDropdown;

    // Define the sign for each language
    public GameObject LIFTItalian;
    public GameObject LIFTLithuanian;
    public GameObject LIFTUrdu;
    public GameObject FALLItalian;
    public GameObject FALLLithuanian;
    public GameObject FALLUrdu;

    void Start()
    {
        // Create new event - On dropdown value change
        myDropdown.onValueChanged.AddListener(delegate {
            myDropdownValueChangedHandler(myDropdown);
        });
    }

    void Destroy()
    {
        // Destroy event as soon as user quits the app
        myDropdown.onValueChanged.RemoveAllListeners();
    }
```

Figura 7.57: Script translate pt1

```
private void myDropdownValueChangedHandler(Dropdown target)
{
    // If "English" is selected
    if (target.value == 0)
    {
        // Hide translated signage
        LIFTItalian.SetActive(false);
        LIFTLithuanian.SetActive(false);
        LIFTUrdu.SetActive(false);
        FALLItalian.SetActive(false);
        FALLLithuanian.SetActive(false);
        FALLUrdu.SetActive(false);
    }

    // If "Italian" is selected
    if (target.value == 1)
    {
        // Hide non-Italian signage
        // (The English sign is the marker itself,
        //thus the Italian signage will be displayed as an overlay)
        LIFTItalian.SetActive(true);
        LIFTLithuanian.SetActive(false);
        LIFTUrdu.SetActive(false);
        FALLItalian.SetActive(true);
        FALLLithuanian.SetActive(false);
        FALLUrdu.SetActive(false);
    }

    // If "Lithuanian" is selected
    if (target.value == 2)
    {
        // Hide non-Lithuanian signage
        LIFTItalian.SetActive(false);
        LIFTLithuanian.SetActive(true);
        LIFTUrdu.SetActive(false);
        FALLItalian.SetActive(false);
        FALLLithuanian.SetActive(true);
        FALLUrdu.SetActive(false);
    }

    // If "Urdu" is selected
    if (target.value == 3)
    {
        // Hide non-Urdu signage
        LIFTItalian.SetActive(false);
        LIFTLithuanian.SetActive(false);
        LIFTUrdu.SetActive(true);
        FALLItalian.SetActive(false);
        FALLLithuanian.SetActive(false);
        FALLUrdu.SetActive(true);
    }
}

// Set the index for each value in the dropdown
public void SetDropdownIndex(int index)
{
    myDropdown.value = index;
}
```

Figura 7.58: Script translate pt2

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Vuforia;

public class AutoFocus : MonoBehaviour {

    void Start()
    {

        // Get the Vuforia instance of the app
        var vuforia = VuforiaARController.Instance;

        // Register OnVuforiaStarted event
        vuforia.RegisterVuforiaStartedCallback(OnVuforiaStarted);

        // Register OnPaused event
        vuforia.RegisterOnPauseCallback(OnPaused);
    }

    // As soon as Vuforia is initialized
    private void OnVuforiaStarted()
    {

        // Override FocusMode
        CameraDevice.Instance.SetFocusMode(

            // Enable continuos autofocus
            CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
    }

    // When Vuforia is paused
    private void OnPaused(bool paused)
    {
        // When it's resumed
        if (!paused)
        {
            // Focus mode may be reset to default
            // Therefore, re-enable autofocus
            CameraDevice.Instance.SetFocusMode(
                CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
        }
    }
}
```

Figura 7.59: Script AutoFocus

7.2.1 Import dell'applicazione

Dopo aver scaricato da GitHub il file zip contenente il progetto, è stato estratto e importato in UnityHub selezionando l'apposita versione di Unity (2018.4.30f1). Una volta averlo aperto con Unity, è stato necessario inserire la chiave personale per attivare le funzioni di Vuforia. In seguito, è stato importato Poco-SDK all'interno del progetto effettuando i passi mostrati nel paragrafo 4.5.3. In questo modo, sarà possibile accedere alla gerarchia di oggetti per effettuare un'implementazione più precisa dello script di test. Invece per quanto riguarda la creazione del file di log, necessario per la valutazione della copertura dei rami, al progetto è stato aggiunto lo script "SondaManager", mostrato nella sezione 6.3.1, e a ognuno degli script sono state aggiunte le sonde seguendo la metodologia proposta, anch'essa, nella sezione 6.3.1. Di seguito sono mostrati gli script del progetto che, rispetto a quelli mostrati nelle figure 7.60, 7.61, 7.62 e 7.63 presentano l'aggiunta delle suddette sonde. In fine, l'applicazione è stata deployata sul dispositivo simulato mostrato nei capitoli precedenti.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class playbackspeed : MonoBehaviour
{
    //Define Animation component
    public Animation anim;

    void Start()
    {
        SondaManager.inserisciSonda("playbackspeed.Start");
        // Set playback speed of "lift" animation to 0.25 (floating point)
        anim["lift"].speed = 0.25f;
    }
}
```

Figura 7.60: Script playbackspeed con sonde

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class translate : MonoBehaviour
{
    // Define the dropdown GameObject
    public Dropdown myDropdown;

    // Define the sign for each language
    public GameObject LIFTItalian;
    public GameObject LIFTLithuanian;
    public GameObject LIFTUrdu;
    public GameObject FALLItalian;
    public GameObject FALLLithuanian;
    public GameObject FALLUrdu;

    void Start()
    {
        SondaManager.inserisciSonda("translate.Start");
        // Create new event - On dropdown value change
        myDropdown.onValueChanged.AddListener(delegate {
            myDropdownValueChangedHandler(myDropdown);
        });
    }

    void Destroy()
    {
        SondaManager.inserisciSonda("translate.Destroy");
        // Destroy event as soon as user quits the app
        myDropdown.onValueChanged.RemoveAllListeners();
    }
```

Figura 7.61: Script translate pt1 con sonde

```
    // When dropdown value changes
    private void myDropdownValueChangedHandler(Dropdown target)
    {
        SondaManager.inserisciSonda("translate.myDropdownValueChangedHandler");
        // If "English" is selected
        if (target.value == 0)
        {
            SondaManager.inserisciSonda("translate.myDropdownValueChangedHandler.if (target.value == 0)");
            // Hide translated signage
            LIFTItalian.SetActive(false);
            LIFTLithuanian.SetActive(false);
            LIFTUrdu.SetActive(false);
            FALLItalian.SetActive(false);
            FALLLithuanian.SetActive(false);
            FALLUrdu.SetActive(false);
        }

        // If "Italian" is selected
        if (target.value == 1)
        {
            SondaManager.inserisciSonda("translate.myDropdownValueChangedHandler.if (target.value == 1)");
            // Hide non-Italian signage
            // (The English sign is the marker itself, thus the Italian signage will be displayed as an overlay)
            LIFTItalian.SetActive(true);
            LIFTLithuanian.SetActive(false);
            LIFTUrdu.SetActive(false);
            FALLItalian.SetActive(true);
            FALLLithuanian.SetActive(false);
            FALLUrdu.SetActive(false);
        }

        // If "Lithuanian" is selected
        if (target.value == 2)
        {
            SondaManager.inserisciSonda("translate.myDropdownValueChangedHandler.if (target.value == 2)");
            // Hide non-Lithuanian signage
            LIFTItalian.SetActive(false);
            LIFTLithuanian.SetActive(true);
            LIFTUrdu.SetActive(false);
            FALLItalian.SetActive(false);
            FALLLithuanian.SetActive(true);
            FALLUrdu.SetActive(false);
        }

        // If "Urdu" is selected
        if (target.value == 3)
        {
            SondaManager.inserisciSonda("translate.myDropdownValueChangedHandler.if (target.value == 3)");
            // Hide non-Urdu signage
            LIFTItalian.SetActive(false);
            LIFTLithuanian.SetActive(false);
            LIFTUrdu.SetActive(true);
            FALLItalian.SetActive(false);
            FALLLithuanian.SetActive(false);
            FALLUrdu.SetActive(true);
        }
    }

    // Set the index for each value in the dropdown
    public void SetDropdownIndex(int index)
    {
        SondaManager.inserisciSonda("translate.SetDropdownIndex");
        myDropdown.value = index;
    }
}
```

Figura 7.62: Script translate pt2 con sonde

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Vuforia;

public class AutoFocus : MonoBehaviour {

    void Start()
    {
        SondaManager.inserisciSonda("AutoFocus.Start");

        // Get the Vuforia instance of the app
        var vuforia = VuforiaARController.Instance;

        // Register OnVuforiaStarted event
        vuforia.RegisterVuforiaStartedCallback(OnVuforiaStarted);

        // Register OnPaused event
        vuforia.RegisterOnPauseCallback(OnPaused);
    }

    // As soon as Vuforia is initialized
    private void OnVuforiaStarted()
    {
        SondaManager.inserisciSonda("AutoFocus.OnVuforiaStarted");

        // Override FocusMode
        CameraDevice.Instance.SetFocusMode(

            // Enable continuos autofocus
            CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
    }

    // When Vuforia is paused
    private void OnPaused(bool paused)
    {
        SondaManager.inserisciSonda("AutoFocus.OnPaused");
        // When it's resumed
        if (!paused)
        {
            SondaManager.inserisciSonda("AutoFocus.OnPaused.if (!paused)");
            // Focus mode may be reset to default
            // Therefore, re-enable autofocus
            CameraDevice.Instance.SetFocusMode(
                CameraDevice.FocusMode.FOCUS_MODE_CONTINUOUSAUTO);
        }
    }
}
```

Figura 7.63: Script AutoFocus con sonde

7.2.2 Modellazione

Seguendo ciò che è stato teorizzato nella sezione 6.2.4, si procede con la modellazione di "PointAR". Così come "Safari Animal AR", anche l'ap-

plicazione in esame non presenta alcun SRS, tuttavia le informazioni relative al suo funzionamento possono essere reperite dalla descrizione presente sul sito [25] e su Google Play Store. Tali informazioni sono state utilizzate per la stesura della sezione 7.2, utile per definire il comportamento dell'applicazione. Seguendo le nozioni presentate nella sezione 6.2.2, si effettua la modellazione dell'applicazione in esame. In prima battuta è necessario identificare gli event e le conseguenti action di tale applicazione. La tabella 7.25 mostra quelli identificati dalla descrizione di "PointAR".

Event	**Action**
Selezione inglese da menù a tendina	Selezione traduzione inglese da mostrare
Selezione italiano da menù a tendina	Selezione traduzione italiana da mostrare
Selezione lituano da menù a tendina	Selezione traduzione lituana da mostrare
Selezione urdu da menù a tendina	Selezione traduzione urdu da mostrare
Click bottone Start	Attivazione fotocamera
Riconoscimento marker 1	Creazione relativa scena AR tradotta
Disconoscimento marker 1	Scomparsa Relativa scena AR tradotta
Riconoscimento marker 2	Creazione relativa scena AR tradotta
Disconoscimento marker 2	Scomparsa Relativa scena AR tradotta
Click bottone ingranaggio	Comparsa menù a tendina e bottone "Resume"
Click bottone Resume	Attivazione fotocamera

Tabella 7.25: Tabella event e action

Si passa a definire il numero di stati atti a descrivere tale applicazione. Per fare ciò è necessario applicare la nozione di invariante di stato e post-condizioni. Per l'invariante di stato si hanno le seguenti condizioni:

- verificare la possibilità di selezionare la lingua;
- verificare l'assenza della scena;
- verificare la comparsa della scena 1 tradotta;
- verificare la comparsa della scena 2 tradotta.

Mentre per le post condizioni si hanno le situazioni espresse nella colonna delle action. Pertanto gli stati riconoscibili sono i seguenti:

- Seleziona lingua: verifica che sia possibile selezionare la lingua per la traduzione;
- Attesa: verifica l'assenza delle scene AR;
- Sollevamento: verifica che sia creata la scena di realtà aumentata relativa al marker 1;
- Caduta: verifica che sia creata la scena di realtà aumentata relativa al marker 2.

Ottenuti gli stati si passa alla determinazione delle transition applicando l'omonima nozione, descritta nella sezione 6.2.2, la quale afferma

che a ogni event, presente in tabella 7.25, deve essere associata almeno una transition. La FSM che ne scaturisce è mostrata di seguito.

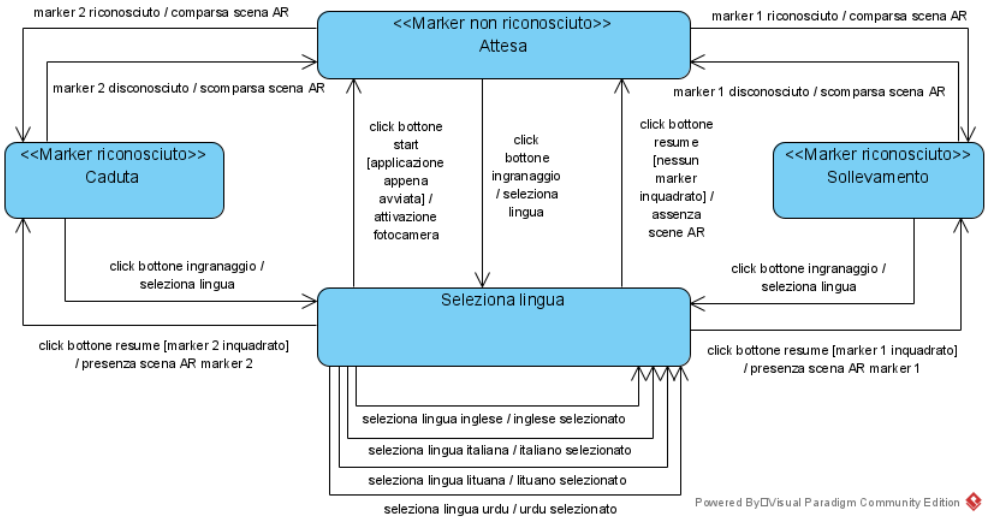


Figura 7.64: FSM PointAR

Dall'immagine si può evincere che: all'event legato al click del bottone "Resume" sono associate tre transition e lo stesso dicasi per l'evento di click sul bottone dell'ingranaggio. Inoltre alcune transition presentano delle condizioni di guardia che ne determinano l'attivazione. Più nel dettaglio si ha:

- La transition del click del bottone "Start" presenta la condizione di guardia che afferma che esso risulta essere cliccabile solo all'avvio dell'applicazione;
- La transition relativa al click del bottone "Resume", che porta da "Seleziona lingua" ad "Attesa", presenta la condizione di guardia per la quale è reso possibile il suo click solo se non si tratta dell'avvio dell'applicazione e non è presente alcun marker;

- La transition relativa al click del bottone "Resume", che porta da "Seleziona lingua" a "Sollevamento", presenta la condizione di guardia per la quale è reso possibile il suo click solo se non si tratta dell'avvio dell'applicazione ed è presente il marker 1;

- La transition relativa al click del bottone "Resume", che porta da "Seleziona lingua" a "Caduta", presenta la condizione di guardia per la quale è reso possibile il suo click solo se non si tratta dell'avvio dell'applicazione ed è presente il marker 2;

Anche in questo caso è possibile semplificare il modello mostrato in figura 7.64, infatti è possibile parametrizzare le transition relative alla scelta della lingua. Di seguito si mostra la FSM risultante.

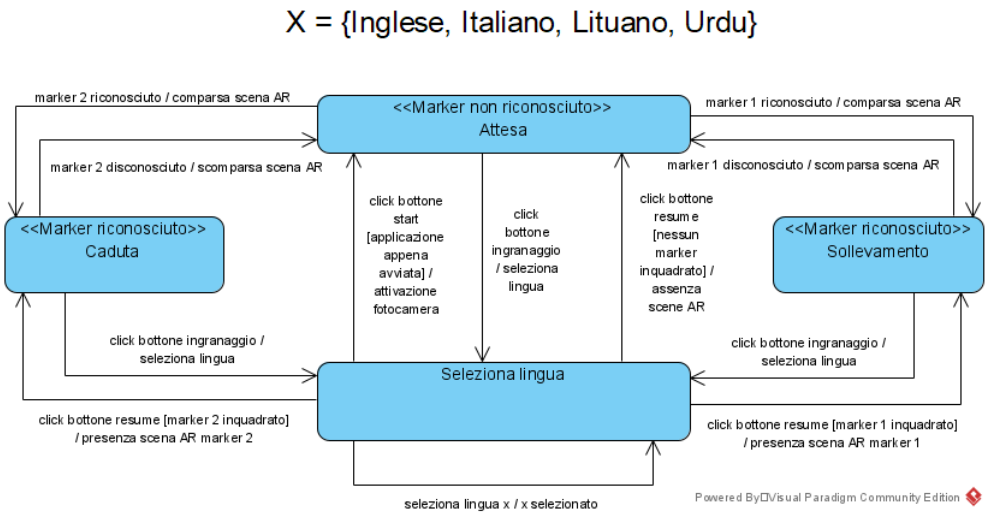


Figura 7.65: FSM parametrizzata PointAR

Nella figura soprastante è stata effettuata la seguente semplificazione: i quattro cappi di "Seleziona lingua" sono stati sostituiti da un unico cappio parametrizzato, dove x rappresenta la variabile appar-

tenente al dominio X, discreto e finito, contenente le diverse lingue selezionabili. Chiaramente, ogni volta che si sceglie di utilizzare tale transizione, la variabile x può assumere solo uno dei valori del dominio X. Per soddisfare la nozione di stato iniziale e finale, alla FSM di figura 7.65 sono stati aggiunti gli pseudo-stati iniziale e finale con le relative transizioni. Nel caso specifico, sarà presente una sola transizione che parte dal pseudo-stato iniziale e giunge allo stato "Seleziona lingua" e rappresenta l'event di avvio dell'applicazione (omesso nella tabella 7.25); mentre il pseudo-stato finale avrà quattro transizioni in ingresso, aventi origine da ognuno degli altri stati (ad esclusione del pseudo-stato iniziale) e rappresentanti la chiusura dell'applicazione (anch'essa omessa nella tabella 7.25. La FSM risultante è mostra di seguito. Si passa ora a validare la nozione di Reachability, verificando che ogni stato, escludendo quello iniziale e finale, abbia almeno una transizione in ingresso e una in uscita. Come si può evincere dall'immagine, tale proprietà è soddisfatta. In ultimo si verifica che la FSM costruita sia minima, ossia non presenti stati equivalenti. Per accertarsi di ciò si utilizza la definizione presentata nella sezione 6.2.2.

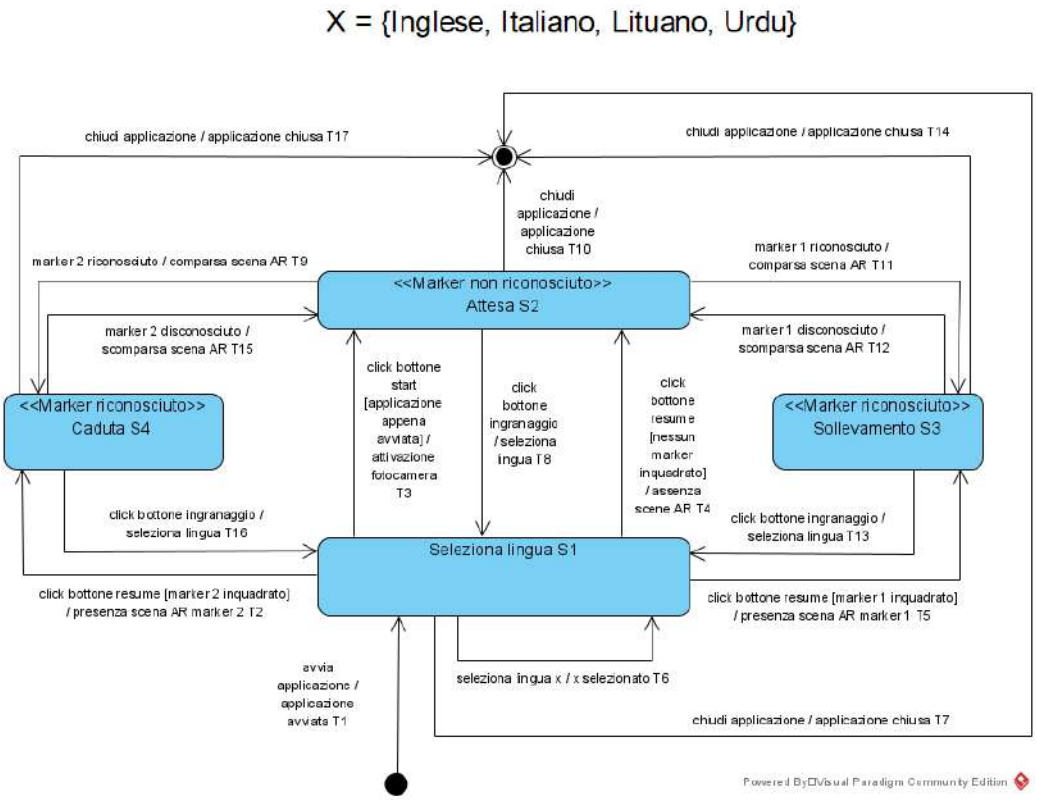


Figura 7.66: FSM completa PointAR

Nel caso in esame sono presenti quattro stati, escludendo quello iniziale e finale, quindi le coppie di stati legate da almeno una transizione sono:

- "Seleziona lingua" - "Attesa", tale coppia presenta tre transizioni due da "Seleziona lingua" verso "Attesa" e una da "Attesa" verso "Seleziona lingua". Si può quindi concludere che i due stati non producono lo stesso comportamento;
- "Attesa" - "Sollevamento", tale coppia presenta solo due transizioni di riconoscimento e disconoscimento del marker che portano rispettivamente da "Attesa" a "Sollevamento" e viceversa. Di conseguenza il comportamento prodotto è differente;

- "Attesa" - "Caduta", si può applicare lo stesso ragionamento fatto in precedenza. Dato che anche questa coppia presenta solo le due transizioni di avvenuto riconoscimento e disconoscimento del marker;

- "Seleziona lingua" - "Sollevamento", anche questa coppia presenta solo due transizioni e applicando il ragionamento precedente si definisce il loro comportamento differente;

- "Seleziona lingua" - "caduta", ugualmente alla precedente presenta due transizioni e si conclude che il comportamento è differente.

Da quanto detto si può sancire che gli stati non sono fra essi equivalenti e la FSM di figura 7.66 risulta essere minima. Si noti che tale FSM presenta gli stati e le transizioni numerate, per meglio capire a quali di essi ci si riferirà in ogni test case.

7.2.3 Definizione della test suite

Dopo aver creato il modello nella sezione precedente, si passa a scaturire da esso la test suite. Per fare ciò sarà necessario applicare i criteri di copertura mostrati nella sezione 6.2.3. Anche in questo caso il criterio di copertura All-Paths Coverage sarà scartato poiché il modello creato presenta loop e cappi e, per tale ragione, i paths possibili risulterebbero essere infiniti. Per quanto riguarda gli altri criteri saranno tutti applicati al fine di confrontarli e definire quali tra essi risulta essere il

migliore per il testing dell'applicazione in esame. Per ognuno dei criteri sarà creata la tabella, presentata nella sezione 6.2.5, contenente i test case da effettuare. Si noti che tra i vari test case, nessuno è atto a verificare la correttezza del testo presente sul cartello. Ciò è dovuto al fatto che il cartello tradotto che compare non è altro che un'immagine e come tale non è possibile accedere al testo.

Test suite di All-States Coverage

La test suite scaturente da tale criterio ha il fine di coprire almeno una volta tutti gli stati, quindi si pone l'obiettivo di individuare i paths che permettano quanto detto. Come nel caso di "Safari Animal AR", essendo che la FSM di "PointAR" presenta cappi e loop i possibili paths potrebbero essere potenzialmente infiniti. Per eludere tale problematica si impone il vincolo che ogni path possa presentare al più una volta ogni stato della FSM, soddisfacendo così il requisito minimo del criterio scelto. Omettendo il pseudo-stato iniziale, lo stato di partenza dell'applicazione sarà "Seleziona lingua" in cui si tralascerà il cappio parametrizzato, dato che porterebbe nuovamente nel medesimo stato violando il vincolo precedentemente imposto, e di conseguenza l'unica transizione percorribile risulterà essere il click del tasto "Start". Tale transizione sarà l'unica selezionabile poiché le altre, relative al click del bottone "Resume", presentano la condizione di guardia che non le rende percorribili appena avviata l'applicazione. In definitiva,

scegliendo la transizione relativa al click del bottone "Start", si passa dallo stato "Seleziona lingua" ad "Attesa". Da quest'ultimo non potrà essere scelta la transizione relativa al click dell'ingranaggio, poiché porterebbe nuovamente nello stato "Seleziona lingua", precedentemente visitato. Di conseguenza le uniche transizioni percorribili risultano essere quelle relative al riconoscimento del marker 1 e del marker 2. Ovviamente se ne potrà scegliere esclusivamente una. Supponendo di scegliere quella relativa al riconoscimento del marker 1, si passerà dallo stato "Attesa" allo stato "Sollevamento". Da qui qualsiasi transizione si scegliesse porterebbe in uno stato già visitato, quindi il primo path può definirsi concluso. Nonostante sia stato individuato tale path non sono stati coperti tutti gli stati, quindi è necessario trovarne un altro che permetta la copertura dell'unico stato mancante, "Caduta". Si può intuire che trovare tale path è abbastanza semplice, infatti saranno effettuate esattamente le stesse scelte del previo path al fine di arrivare allo stato "Attesa". Da qui, differentemente dal precedente, sarà scelta la transizione di riconoscimento del marker 2, in modo tale da percorrerla e giungere nello stato "Caduta". Arrivati in tale stato qualsiasi transizione si scelga porterà in uno stato già visitato e, per tale ragione, anche questo path può dichiararsi concluso. Nella tabelle 7.26 e 7.27, si mostra la test suite risultante da tale criterio.

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
1	Click del bottone "Start"	Applicazione avviata	Attivazione fotocamera	S1, S2	T3
2	Comparsa scena AR del marker 1	Inquadramento marker 1	Comparsa relativo cartello tradotto e animazione	S2, S3	T11

Tabella 7.26: Tabella test suite All-States Coverage path 1

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
1	Click del bottone "Start"	Applicazione avviata	Attivazione fotocamera	S1, S2	T3
3	Comparsa scena AR del marker 2	Inquadramento marker 2	Comparsa relativo cartello tradotto e animazione	S2, S4	T9

Tabella 7.27: Tabella test suite All-States Coverage path 2

Si noti che il test avente ID pari a 1 sarà utilizzato in entrambi i paths poiché con esso si effettua il passaggio dallo stato "Seleziona lingua" ad "Attesa". Di seguito si riporta la tabella contenente la statistica relativa alla percentuale di copertura degli stati e delle transizioni.

	Coperti	Totali	Percentuale
Stati	4	4	100 %
Transizioni	3	15	20 %

Tabella 7.28: Percentuale copertura Stati e Transizioni All-States Coverage

Si noti che il numero di transizione è dato dalla somma di tutte le transizioni a cui vanno aggiunte tutte le possibili declinazioni del cappio parametrizzato. Come era lecito aspettarsi, la copertura degli stati è totale contrariamente rispetto alle transizioni, coperte solo in minima parte. Di seguito si riporta la tabella contenete il numero di test case che è necessario effettuare per tale test suite.

N° test case
4

Tabella 7.29: N° test case All-States Coverage

Test suite di All-Transitions Coverage

Tale criterio ha lo scopo di coprire tutte le possibili transizioni almeno una volta. Anche in questo caso saranno tralasciati gli pseudo-stati iniziale e finale con le relative transizioni. Invece, per ognuno dei rimanenti stati sarà necessario verificare le transizione ad esso associate. Di seguito si mostra la test suite relativa a tale criterio.

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
1	Selezione lingua inglese	Applicazione avviata	Lingua inglese selezionata	S1	T6 (x = inglese)
2	Selezione lingua italiana	Applicazione avviata	Lingua italiana selezionata	S1	T6 (x = italiano)
3	Selezione lingua lituana	Applicazione avviata	Lingua lituana selezionata	S1	T6 (x = lituano)
4	Selezione lingua urdu	Applicazione avviata	Lingua urdu selezionata	S1	T6 (x = Urdu)
5	Click del bottone “Start”	Applicazione avviata	Attivazione fotocamera	S1, S2	T3
6	Possibilità di selezionare nuovamente una lingua	Fotocamera attivata	Possibilità di scegliere una lingua	S2, S1	T8
7	Click tasto “Resume”	Sostituzione lingua precedentemente scelta	Attivazione fotocamera	S1, S2	T4
8	Comparsa scena AR del marker 1	Inquadramento marker 1	Comparsa relativo cartello tradotto e animazione	S2, S3	T11
9	Possibilità di selezionare nuovamente una lingua	Marker 1 riconosciuto	Possibilità di scegliere una lingua	S3, S1	T13

ID	Descrizione	Pre-Condizione	Post-Condizione	Stati	Transizioni
10	Click tasto "Resume"	Sostituzione lingua precedentemente scelta	Ritorno alla scena AR relativa al marker 1	S1, S3	T5
11	Scomparsa scena AR del marker 1	Marker 1 riconosciuto	Scomparsa scena AR relativa al marker 1	S3, S2	T12
12	Comparsa scena AR del marker 2	Inquadramento marker 2	Comparsa relativo cartello tradotto e animazione	S2, S4	T9
13	Possibilità di selezionare nuovamente una lingua	Marker 2 riconosciuto	Possibilità di scegliere una lingua	S4, S1	T16
14	Click tasto "Resume"	Sostituzione lingua precedentemente scelta	Ritorno alla scena AR relativa al marker 2	S1, S4	T2
15	Scomparsa scena AR del marker 2	Marker 2 riconosciuto	Scomparsa scena AR relativa al marker 2	S4, S2	T15

Tabella 7.30: Tabella test suite All-Transitions Coverage

Si noti che sono presenti test per ognuno dei valori del cappio parametrizzato. Da tale test suite ciò che scaturisce è non solo la copertura di tutte le transizioni, ma anche la copertura di tutti gli stati. Quan-

to detto è mostrato nella tabella 7.31, che presenta le percentuali di copertura riguardanti stati e transizioni.

	Coperti	**Totali**	**Percentuale**
Stati	4	4	100 %
Transizioni	15	15	100 %

Tabella 7.31: Percentuale copertura Stati e Transizioni All-Transiotions Coverage

Di seguito si riporta la tabella contenete il numero di test case che è necessario effettuare per tale test suite.

N° test case
15

Tabella 7.32: N° test case All-Transitions Coverage

All-One-Loop-Paths Coverage

Lo scopo di tale criterio è individuare tutti i possibili paths aventi uno stato ripetuto. Per la costruzione di tali paths, si utilizzeranno tutti gli stati del modello di figura 7.66, ad esclusione degli pseudo-stati iniziale e finale. Come è possibile intuire, i paths validi possono avere lunghezze differenti, di seguito si mostra come sono stati individuati. Avendo escluso il pseudo-stato iniziale, lo stato di partenza sarà "Seleziona lingua", quindi si può facilmente intuire che è possibile avere 4 paths validi di lunghezza due, grazie ai quattro valori del cappio parametrizzato in esso presente. Si passa quindi a valutare i paths validi

di lunghezza tre, partendo da quelli di lunghezza due precedentemente trovati. Infatti aggiungendo ad ognuno di essi lo stato "Attesa", mediante l'unica transizione possibile (relativa al click del bottone "Start") si ottengono altri 4 paths validi. Oltre a questi, va considerato il caso in cui dallo stato "Seleziona lingua" si passi direttamente allo stato di "Attesa" e da questo si torni indietro mediante la transizione di click sull'ingranaggio. Tale transizione risulta essere l'unica a generare un path valido, dato che le transizioni di riconoscimento dei marker porterebbero nei rispettivi stati, creando paths senza ripetizioni. In definitiva il totale dei paths si lunghezza tre è pari a 5. Si procede nella valutazione di paths validi di lunghezza quattro e per farlo si utilizzano i paths di lunghezza tre succitati. Considerando i quattro path di lunghezza tre, contenenti la ripetizione dello stato "Seleziona lingua" nelle prime due posizioni, si valuta se dallo stato "Attesa" è possibile raggiungere altri stati non visitati in precedenza. Ciò è possibile mediante le due transizioni di riconoscimento del marker 1 e del marker 2, che portano nei rispettivi stati. Quindi i paths validi che scaturiscono sono 8 (dati dal prodotto dei 4 paths di lunghezza tre moltiplicati per le 2 transizioni di riconoscimento del marker). A questo punto si passa al rimanente path di lunghezza tre ("Seleziona lingua" - "Attesa" - "Seleziona lingua"), valutando se da esso è possibile scaturire path validi di lunghezza quattro. Trovandoci nello stato "Seleziona lingua" sono percorribili le tre transizioni relative al click

del bottone "Resume". Tuttavia quella che porta da "Seleziona lingua" ad "Attesa" non può essere scelta, dato che condurrebbe in uno stato già visitato e di conseguenza avremmo un path contenete due stati ripetuti, violando il vincolo imposto dal criterio che si sta adoperando. Invece, le altre due transizioni, relative al click del bottone "Resume", possono essere percorse poiché portano in stati non ancora visitati. In questo modo sono stati trovati altri 2 paths validi di lunghezza 4. Ora si valuta se oltre a questi paths è possibile trovarne altri aventi stessa lunghezza. Si parte dallo stato "Seleziona lingua" e mediante l'unica transizione percorribile (a causa delle condizioni di guardia), si giunge nello stato "Attesa". In esso si considerano le transizioni relative al riconoscimento del marker 1 e del marker 2, che portano rispettivamente nello stato "Sollevamento" e "Caduta". Supponendo di percorrere come prima transizione quella relativa al marker 1, si giunge allo stato "Sollevamento". In esso sono possibili due transizioni che generano path validi, in particolare: la transizione relativa al disconoscimento del marker 1 e quella relativa al click dell'ingranaggio che creano paths validi rispettivamente con la ripetizione dello stato "Attesa" e "Seleziona lingua". Ragionamento analogo si può fare se dallo stato "Attesa", anziché scegliere la transizione di riconoscimento del marker 1, si scelga la transizione di riconoscimento del marker 2. Infatti, dallo stato "Caduta" si hanno a disposizione le transizioni di disconoscimento del marker 2 e di click dell'ingranaggio, che producono un comportamen-

to uguale a quello descritto per lo stato "Sollevamento". In definitiva i paths validi di lunghezza quattro risultano essere 14. Si prosegue esaminando la presenza di paths validi di lunghezza 5. Si verifica se è possibile scaturire tali paths da quelli di lunghezza quattro trovati in precedenza. Gli unici paths che si sono dimostrati validi risultano essere i seguenti:

- "Seleziona lingua" - "Attesa" - "Sollevamento" - "Seleziona lingua", infatti trovandosi nello stato "Seleziona lingua" l'unica transizione possibile che può essere effettuata, per non tornare in uno stato già visitato, è quella del click del tasto "Resume" con la guardia relativa alla presenza del marker 2;
- "Seleziona lingua" - "Attesa" - "Caduta" - "Seleziona lingua", è possibile fare un ragionamento analogo al precedente con l'unica differenza di scegliere la transizione del click del tasto "Resume" con la guardia relativa alla presenza del marker 1;

Quindi i paths di lunghezza 5 validi risultano essere 2. In definitiva il numero totale di paths validi identificati è pari a 25, un numero molto più alto rispetto alle test suite precedenti. Infatti si pensi che, per ogni path, è necessario un test case per ogni stato in esso presente. Per tale ragione si evita di riportare le tabelle di tale test suite. Nella tabella 7.33 sono riportate le percentuali di copertura di stati e transizioni.

	Coperti	Totali	Percentuale
Stati	4	4	100 %
Transizioni	15	15	100 %

Tabella 7.33: Percentuale copertura Stati e Transizioni All-One-Loop-Paths Coverage

Si noti che, anche se non sono state riportate le tabelle della test suite di tale criterio, è facile ottenere questi numeri semplicemente analizzando i percorsi precedentemente definiti. Di seguito si riporta la tabella contenete il numero di test case che è necessario effettuare per tale test suite.

N° test case
64

Tabella 7.34: N° test case All-One-Loop-Paths Coverage

7.2.4 Implementazione ed esecuzione script di test

Per l'esecuzione degli script di test che saranno mostrati, è stato necessario avviare il dispositivo emulato, su cui era installato "PointAR", collegandolo ad AirTestIDE tramite ADB. Nell'attuale caso di studio, diversamente dal precedente, sono presenti due marker che saranno posizionati, in modo esclusivo, sul muro della stanza simulata. Una condizione che bisogna rispettare è di inquadrare i marker nella corretta orientazione, come mostrato nell'immagine 7.54, al fine di essere

sicuri di visualizzare le scene di realtà aumenta nel modo corretto. Se per il precedente caso di studio si è utilizzato il dispositivo alternativo, descritto al link [24], è necessario fare lo stesso con il corrente caso in esame. Per poter accedere alla gerarchia di oggetti dell'applicazione è necessario selezionare "Unity" nella finestra Poco di AirTestIDE. Ciò ha permesso di capire come gli oggetti fossero stati organizzati dallo sviluppatore e su quali di essi era necessario concentrarsi per effettuare i tests.

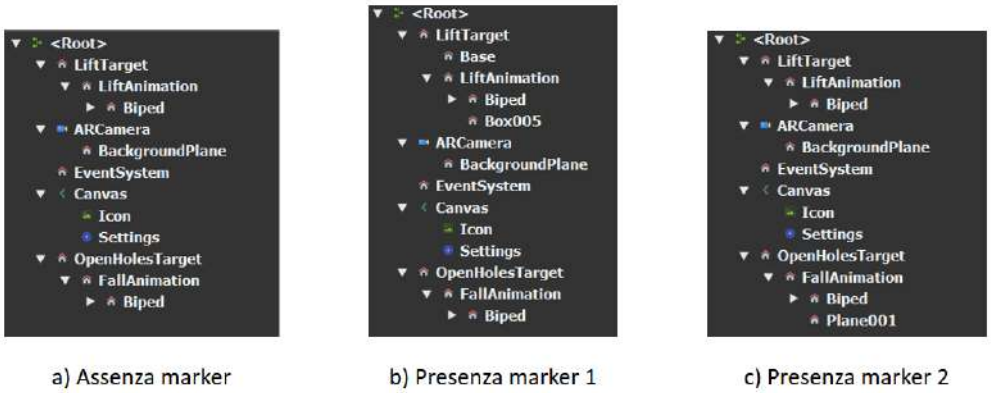


Figura 7.67: Gerarchia oggetti

Si è notato che gli oggetti relativi alle animazioni delle scene AR, nonostante nessun marker fosse inquadrato, erano comunque istanziati. Per questa ragione, l'attenzione si è spostata sul cubo da sollevare e sulla piattaforma utilizzata per camminare, rispettivamente oggetti relativi al marker 1 e al marker 2. Infatti, tali oggetti comparivano esclusivamente quando era inquadrato il relativo marker e per questa ragione, come sarà mostrato nei successivi paragrafi, saranno utilizzati nei tests per verificare la comparsa delle scene AR. Si fa inoltre pre-

sente che lo spostamento del dispositivo mobile, così come il cambio di marker, deve necessariamente essere effettuato a mano. Di conseguenza i test che prevedono tali interazioni non potranno essere effettuati mediante script.

Script test suite All-States Coverage

Diversamente dal precedente caso di studio, sono stati realizzati due script per l'implementazione di tale test suite. Ciò è dovuto al fatto che, come precedentemente anticipato, è necessario un previo cambio di marker con conseguente posizionamento del dispositivo emulato, in un punto tale da inquadrarlo. Nelle figure 7.68 e 7.69 si mostra lo script scaturito dalla test suite presentata nella tabella 7.26. Il path seguito, per l'effettuazione di tale test, è quello ottenuto dall'ordine in cui sono stati inseriti i test case all'interno della suddetta tabella. Lo script si compone di divere funzioni:

- avviaApp: così come quella del caso di studio precedente, ha l'onere di avviare l'applicazione utilizzando i comandi bash ADB. Successivamente è atteso un tempo pari a quindici secondi, necessario per l'avvio dell'applicazione, ed è richiamata la funzione "inizializza". Tale funzione, inizializza la variabile relativa alla lingua di default e la variabile "poco", utile per stabilire la connessione rpc tra l'IDE e il dispositivo emulato;

- chiudiApp: anch'essa simile a quella del caso di studio precedente, ha l'onere di chiudere l'applicazione mediante comandi bash ADB;

- verificaSchermataSelezionaLingua: ha l'onere di verificare che sia correttamente presentata la schermata di selezione della lingua. Il parametro in ingresso è utile a differenziare se si tratta di primo avvio dell'applicazione oppure no, rappresentando così le condizioni di guardia delle transizioni della FSM 7.66 relative al click del bottone "Start" o "Resume". Nello specifico, le istruzioni di tale funzione caricano: tutti i nodi figli del nodo genitore "LanguageSelectionGroup", tutti i bottoni presenti e definiscono il tipo di bottone che deve essere presente a seconda se l'applicazione è stata appena avviata o meno. Fatto ciò si valuta che: il numero di nodi figli di "LanguageSelectionGroup" sia maggiore di zero (rappresenta la presenza del menù a tendina), sia presente esclusivamente un bottone e sia il bottone "Start". Se tutte queste condizioni sono soddisfatte, si è verificato la presenza della scena iniziale;

- verificaScenaAR: ha l'onere di verificare la comparsa della scena AR relativa al marker 1. Ciò è realizzato caricando i nodi figli del nodo genitore "LiftAnimation" e tutti i bottoni presenti sulla scena. Successivamente si verifica che i primi siano superiori a uno (come anticipato, presenza del cubo oltre l'animazione),

che il bottone sia esclusivamente uno e sia il bottone dell'ingranaggio. Se tali condizioni sono soddisfatte si valuta se la lingua selezionata sia quella di default (inglese). Nel caso in cui non lo sia, si verifica la comparsa del marker tradotto nella giusta lingua e, in caso positivo, si verifica la corretta apparizione della scena AR. Se invece la lingua è quella di default non sarà necessario effettuare tale verifica poiché il marker non sarà tradotto essendo già in inglese. Descritti i metodi necessari per lo script, si fa presente che "verificaSchermataSelezionaLingua" è l'oracolo per verificare la correttezza dello stato "Seleziona lingua", mentre "verificaScenaAR" è l'oracolo per verificare la correttezza dello stato "Sollevamento" (scena AR relativa al primo marker). Detto ciò, si passa a definirne l'ordine dell'invocazione delle funzioni sopraccitate. Ovviamente è necessario avviare l'app, verificare la schermata iniziale, cliccare il bottone "Start", verificare la comparsa della scena AR relativa al marker 1 e infine chiudere l'applicazione.

```python
# -*- encoding=utf8 -*-
__author__ = "Danilo"

from airtest.core.api import *
auto_setup(__file__)
from poco.drivers.unity3d import UnityPoco
poco = UnityPoco()

def avviaApp():
    os.system('cmd /c "cd '+os.environ['USERPROFILE']+'\\Desktop\\AirtestIDE\\airtest\\core\\android\\static\\adb\\windows & adb shell am start com.abdu.PointAR/com.unity3d.player.UnityPlayerActivity"')
    time.sleep(15)
    inizializzazione();

def chiudiApp():
    keyevent("KEYCODE_APP_SWITCH")
    os.system('cmd /c "cd '+os.environ['USERPROFILE']+'\\Desktop\\AirtestIDE\\airtest\\core\\android\\static\\adb\\windows & adb shell am force-stop com.abdu.PointAR"')
    os.system('cmd /c "cd '+os.environ['USERPROFILE']+'\\Desktop\\AirtestIDE\\airtest\\core\\android\\static\\adb\\windows & adb shell am kill com.abdu.PointAR"')
    time.sleep(5)

def inizializzazione():

    global poco;
    global linguaSelezionata;

    poco = UnityPoco();
    linguaDefault = "English";
    linguaSelezionata = linguaDefault;

def verificaSchermataSelezioneLingua(tipoSchermata):

    global bottone;

    menuTendina = poco("LanguageSelectionGroup").children();
    bottone = poco(type = 'Button');
    presenzaSchermata = False;
    nomeBottone= None;

    if(tipoSchermata == "avvio"):
        nomeBottone = "StartButton";
    else:
        nomeBottone = "ResumeButton";

    if(len(menuTendina) > 0 and len(bottone) == 1 and bottone.attr('name') == nomeBottone):
        presenzaSchermata = True;

    assert_equal(presenzaSchermata,True,"schermata iniziale comparsa");
```

Figura 7.68: Script test All-States Coverage marker 1 pt1

```
def verificaScenaAr():

    global bottone;
    global linguaSelezionata;

    scenaAR = poco("LiftAnimation").children();
    bottone = poco(type = 'Button');
    comparsaScenaAr = False;

    if(len(scenaAR) > 1 and len(bottone) == 1 and bottone.attr('name') == "Settings"):
        if(linguaSelezionata != "English"):
            markerTradotto = poco("LIFT - "+linguaSelezionata);
            comparsaScenaAr = markerTradotto.exists();
        else:
            comparsaScenaAr = True

    assert_equal(comparsaScenaAr,True,"Scena AR comparsa");


poco = None;
bottone = None;
listaLingue = [];
linguaSelezionata = None;

f = open("AllStatesCoverageInterazioniMarker1.txt","w+");

f.write("Unico paths All States Coverage Marker 1 \n");

avviaApp()
verificaSchermataSelezioneLingua("avvio")
bottone.click();
f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
verificaScenaAr()
chiudiApp()
f.write("Test concluso senza errori \n");
```

Figura 7.69: Script test All-States Coverage marker 1 pt2

A primo acchito la chiusura dell'applicazione potrebbe sembrare superflua, tuttavia è necessaria al fine di predisporre il dispositivo per l'esecuzione della successivo script di test, relativo al marker 2. Quest'ultimo risulta essere molto simile al precedente, per questa ragione in figura 7.70 si mostra solo la funzione, oracolo dello stato "Caduta", che presenta differenze. Quest'ultime sono relative al caricamento dei nodi figli (l'animazione e la piattaforma) rispetto al nodo genitore "FallAnimation" e all'eventuale caricamento del marker tradotto. La restante parte del metodo, così come il resto dello script, è identico a quello relativo al marker 1 di figura 7.68 e 7.69.

```
49 ▾ def verificaScenaAr():
50
51       global bottone;
52       global linguaSelezionata;
53
54       scenaAR = poco("FallAnimation").children();
55       bottone = poco(type = 'Button');
56       comparsaScenaAr = False;
57
58 ▾     if(len(scenaAR) > 1 and len(bottone) == 1 and bottone.attr('name') ==
    "Settings"):
59 ▾         if(linguaSelezionata != "English"):
60               markerTradotto = poco("FALL - "+linguaSelezionata);
61               comparsaScenaAr = markerTradotto.exists();
62 ▾         else:
63               comparsaScenaAr = True
64
65       assert_equal(comparsaScenaAr,True,"Scena AR comparsa");
66
```

Figura 7.70: Metodo script test All-States Coverage marker 2

Così come nel caso di studio precedente, anche nei due script appena mostrati sono presenti istruzioni di scrittura su file. In particolare saranno creati due file, uno per ciascuno script. Che mostreranno le interazioni effettuate con l'applicazione. In caso di errore, il file contenente le interazioni non presenterà, nell'ultima riga, la stringa "Test concluso senza errori" e quindi ripercorrendo le azioni si potrà replicare il malfunzionamento. Se invece in entrambi i file delle interazioni sarà presente la stringa precedentemente citata si potrà dedurre che il test non avrà rilevato errori. Come si può vedere dai report tutti gli assert, necessari alle verifiche delle varie funzionalità dell'applicazione, sono stati verificati con successo, così come confermato dai file delle interazioni, conseguenza della verificata correttezza dei possibili stati dell'applicazione.

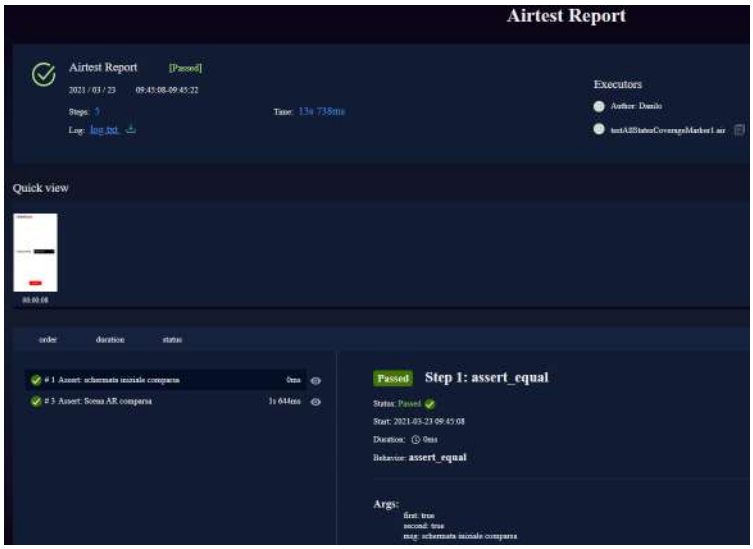


Figura 7.71: Report script test All-States Coverage marker 1

```
Unico paths All States Coverage Marker 1
cliccato bottone: StartButton
Test concluso senza errori
```

Figura 7.72: File interazioni All States Coverage Marker 1

Si noti che si è evitato di riportare le tabelle della test suite poiché la colonna relativa all'esito risulterebbe essere tutta compilata con il valore "passato".

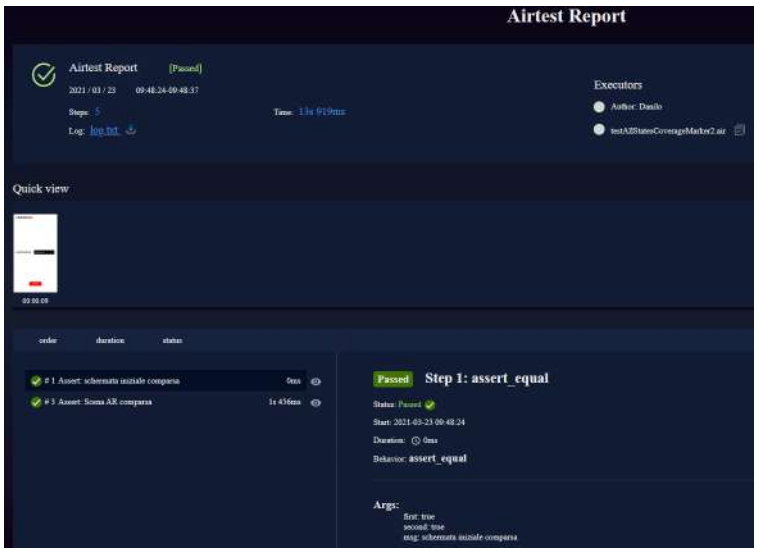


Figura 7.73: Report script test All-States Coverage marker 2

```
Unico paths All States Coverage Marker 2
cliccato bottone: StartButton
Test concluso senza errori
```

Figura 7.74: File interazioni All States Coverage Marker 2

Script test suite All-Transitions Coverage

Anche l'implementazione della test suite mostrata, nella tabella 7.30, ha richiesto la separazione degli script per i motivi sopraccitati. In particolare, sono stati realizzati tre script relativi ai test case riguardanti il marker 1, il marker 2 e l'assenza di entrambi. In altre parole si ha che:

- lo script relativo al marker 1 comprende i test case aventi ID: da 1 a 5 e da 8 a 10;
- lo script relativo al marker 2 comprende i test case aventi ID: 5 e da 12 a 14;
- lo script relativo all'assenza di entrambi i marker comprende i test case aventi ID: da 5 a 7

I rimanenti test case, aventi ID pari a 11 e 15, non è possibile effettuarli dato che, come anticipato, non si è in grado di effettuare il disconoscimento del marker da script. Dopo aver presentato in modo generico i tre script realizzati, si passa a dettagliare come sono stati implementati. Nella figura 7.75 è mostrata la porzione di script al

marker 1. Oltre alle funzioni già citate nel precedente paragrafo, sono presenti:

- apriMenuTendina: ha l'onere di aprire il menù a tendina, con il successivo caricamento di tutti i "toggle" in esso presenti. Quest'ultimi sono gli oggetti che rappresentano le varie lingue selezionabili nel menù a tendina. Successivamente si verifica che il numero di lingue selezionabili sia superiore a zero, ciò rappresenta l'effettiva apertura della menù a tendina poiché le lingue sono istanziate solo all'avvenuta apertura del suddetto menù;

- scorriMenuTendina: ha l'onere di scorrere il menù a tendina per rendere possibile il click sulle lingue che non compaiono per prime.

- selezionaLingua: ha l'onere di selezionare una delle lingue del menù a tendina. Questa funzione ha in ingresso un parametro che rappresenta la lingua da selezionare. Dopo averla caricata si recuperano tutti i suoi nodi figli, in particolare l'ultimo rappresenta la label su cui effettuare il click. Oltre a ciò la lingua selezionata viene salvata per effettuare i futuri controlli riguardanti l'avvenuta e corretta traduzione.

```
69  def apriMenuTendina():
70
71      global listaLingue;
72
73      seleziona = poco("Arrow").click();
74      f.write("cliccato menu tendina \n");
75      listaLingue = poco(type='Toggle');
76      aperturaMenu = False;
77      if(len(listaLingue) > 0):
78          aperturaMenu = True;
79      assert_equal(aperturaMenu,True,"menu a tendina aperto")
80
81  def scorriMenuTendina():
82      poco("Handle").swipe([0.0044, 0.0749]);
83
84  def selezionaLingua(lingua):
85
86      global linguaSelezionata;
87
88      itemDaSelezionare = poco(lingua);
89      itemSelezionato = itemDaSelezionare.children()[2];
90      linguaSelezionata = itemSelezionato.get_text().capitalize();
91      f.write("selezionata lingua: "+linguaSelezionata+"\n");
92      itemSelezionato.click();
93
94
95
96  poco = None;
97  bottone = None;
98  listaLingue = [];
99  linguaSelezionata = None;
100
101 f = open("AllTransitionsCoverageInterazioniMarker1.txt","w+");
102
103 f.write("Unico paths All Transitions Coverage Marker 1 \n");
104
105 avviaApp();
106 verificaSchermataSelezioneLingua("avvio");
107 apriMenuTendina();
108
109 for i in range(len(listaLingue)):
110     if(i == len(listaLingue)-1):
111         scorriMenuTendina();
112     selezionaLingua(listaLingue[i].attr('name'));
113     verificaSchermataSelezioneLingua("avvio");
114     if(i != len(listaLingue)-1):
115         apriMenuTendina();
116
117 bottone.click();
118 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
119 verificaScenaAr();
120 bottone.click();
121 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
122 verificaSchermataSelezioneLingua("AppAvviata");
123 bottone.click();
124 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
125 verificaScenaAr();
126 chiudiApp();
127 f.write("Test concluso senza errori \n");
```

Figura 7.75: Porzione script test All-Transitions Coverage marker

Rispetto alla test suite precedente, si aggiunge un oracolo in più relativo alla verifica della correttezza dello stato “Seleziona lingua”. Tale oracolo risulta essere la funzione “apriMenuTendina”, che verifica

l'effettiva apertura del suddetto menù in caso si scelga come transizione uno dei cappi di "Seleziona lingua". Spiegate le funzioni si passa a dettagliare quale path sia stato seguito per effettuare i test. Sostanzialmente il path seguito è uguale all'ordine dei test case sopra mostrato, infatti gli step seguiti sono:

- avviare l'applicazione e verificare l'avvenuta comparsa della schermata iniziale;
- aprire il menù tante volte quante sono le lingue selezionabili e cliccare ogni volta su una di esse, non precedentemente selezionata;
- cliccare il bottone "Start" e verificare la comparsa della scena AR
- cliccare il bottone ingranaggio e verificare la comparsa della schermata di selezione della lingua, questa volta passando alla funzione "verificaSchermataSelezioneLingua" il parametro che non rappresenta il primo avvio dell'applicazione;
- cliccare il bottone "Rasume", verificare la comparsa della scena AR e chiudere l'applicazione.

Ultimati i chiarimenti inerenti allo script del marker 1 si passa a quello relativo al marker 2. Anche in questo caso il path seguito è lo stesso dell'ordine dei test sopra mostrati. Essendo che lo script di test relativo al marker 2 presenta le stesse funzioni di quello relativo

al marker 1, con le uniche differenze già citate nella sezione 7.2.4, si riporta solo la parte di script relativa all'ordine in cui sono state eseguite.

```
92  poco = None;
93  bottone = None;
94  listaLingue = [];
95  linguaSelezionata = None;
96
97  f = open("AllTransitionsCoverageInterazioniMarker2.txt","w+");
98
99  f.write("Unico paths All Transitions Coverage Marker 2 \n");
100
101 avviaApp();
102 verificaSchermataSelezioneLingua("avvio");
103
104 bottone.click();
105 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
106 verificaScenaAr();
107 bottone.click();
108 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
109 verificaSchermataSelezioneLingua("AppAvviata");
110 bottone.click();
111 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
112 verificaScenaAr();
113 chiudiApp();
114 f.write("Test concluso senza errori \n");
```

Figura 7.76: Script test All-Transitions Coverage marker 2

Come si può vedere dall'immagine 7.76, si è avviata l'applicazione, verificata la comparsa della schermata iniziale, cliccato il bottone "Start", verificata la comparsa della scena AR relativa al marker 2, cliccato il bottone dell'ingranaggio, verificata la comparsa della schermata di selezione lingua, cliccato il bottone "Resume", verificata la comparsa della scena AR e chiusa l'applicazione. Infine, per quanto riguarda lo script relativo all'assenza di entrambe le scene AR, anch'esso presenta le stesse funzioni degli script precedenti tranne "verificaScenaAr". Questa funzione è sostituita con "verificaAssenzaScenaAr", mostrata nell'immagine 7.77, che carica i bottoni presenti nella scena e i nodi fi-

gli dei nodi padri "FallAnimation" e "LiftAnimation". Successivamente verifica che i nodi figli di ognuno dei due nodi padri non sia superiore a uno (come anticipato, presenza esclusiva delle animazioni di ognuno dei marker), che vi sia un solo bottone e che tale bottone sia l'ingranaggio. Se una sola di queste condizione non è rispettata, la verifica dell'assenza delle scene AR fallisce. Come si può intuire tale funzione risulta essere l'oracolo dello stato "Attesa". Riguardo il path si è seguito l'ordine dei test case sopra mostrati.

```
74  def verificaAssenzaScenaAr():
75
76      global bottone;
77      global linguaSelezionata;
78
79      scenaARMarker1 = poco("FallAnimation").children();
80      scenaARMarker2 = poco("LiftAnimation").children();
81      bottone = poco(type = 'Button');
82      comparsaScenaAr = False;
83
84      if((len(scenaARMarker1) > 1 or len(scenaARMarker2) > 1) and len(bottone) == 1
    and bottone.attr('name') == "Settings"):
85          comparsaScenaAr = True
86
87      assert_equal(comparsaScenaAr,False,"Scena AR comparsa");
88
89  poco = None;
90  bottone = None;
91  listaLingue = [];
92  linguaSelezionata = None;
93
94  f = open("AllTransitionsCoverageInterazioniAttesa.txt","w+");
95
96  f.write("Unico paths All Transitions Coverage Attesa \n");
97
98  avviaApp();
99  verificaSchermataSelezioneLingua("avvio");
100
101 bottone.click();
102 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
103 verificaAssenzaScenaAr();
104 bottone.click();
105 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
106 verificaSchermataSelezioneLingua("AppAvviata");
107 bottone.click();
108 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
109 verificaAssenzaScenaAr();
110 chiudiApp();
111 f.write("Test concluso senza errori \n");
```

Figura 7.77: Script test All-Transitions Coverage assenza marker

Dall'immagine 7.77, si può verificare che si è avviata l'applicazio-

ne, verificata la presenza della schermata iniziale, cliccato il bottone "Start", verificata l'assenza delle scene AR, cliccato il bottone dell'ingranaggio, verificata la comparsa della schermata di seleziona lingua, cliccato il bottone "Resume", verificata l'assenza delle scene AR e chiusa l'applicazione. Anche negli script sopraccitati sono presenti le istruzioni di scrittura sul file. Per ognuno degli script sarà creato un file delle interazioni utile per capire le interazioni effettuate in ognuno di essi. Come si può vedere dai report tutti gli assert, necessari alle verifiche delle varie funzionalità dell'applicazione, sono stati verificati con successo, così come confermato dai file delle interazioni. Si noti che si è evitato di riportare la tabella della test suite poiché la colonna relativa all'esito risulterebbe essere tutta compilata con il valore "passato". L'unica eccezione sono i test relativi al disconoscimento del marker che, come anticipato, non si è in grado di eseguirli mediante script.

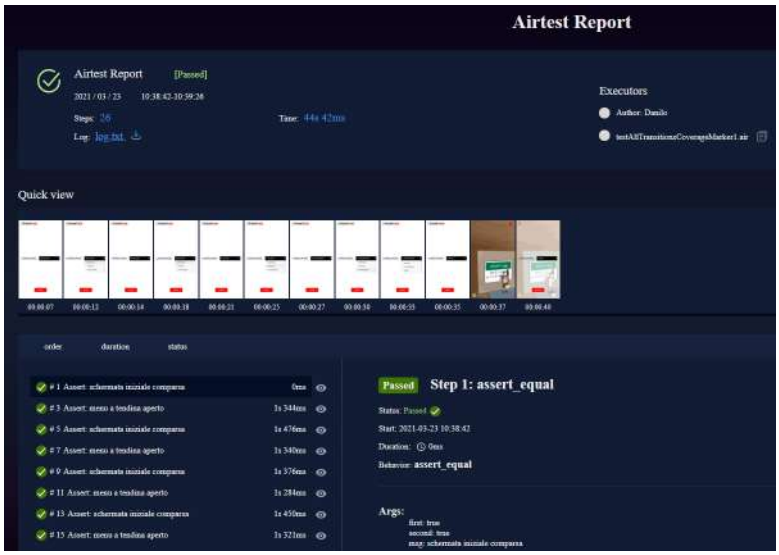


Figura 7.78: Report script test All-Transitions Coverage marker 1

```
Unico paths All Transitions Coverage Marker 1
cliccato menu tendina
selezionata lingua: English
cliccato menu tendina
selezionata lingua: Italian
cliccato menu tendina
selezionata lingua: Lithuanian
cliccato menu tendina
selezionata lingua: Urdu
cliccato bottone: StartButton
cliccato bottone: Settings
cliccato bottone: ResumeButton
Test concluso senza errori
```

Figura 7.79: File interazioni All Transitions Coverage Marker 1

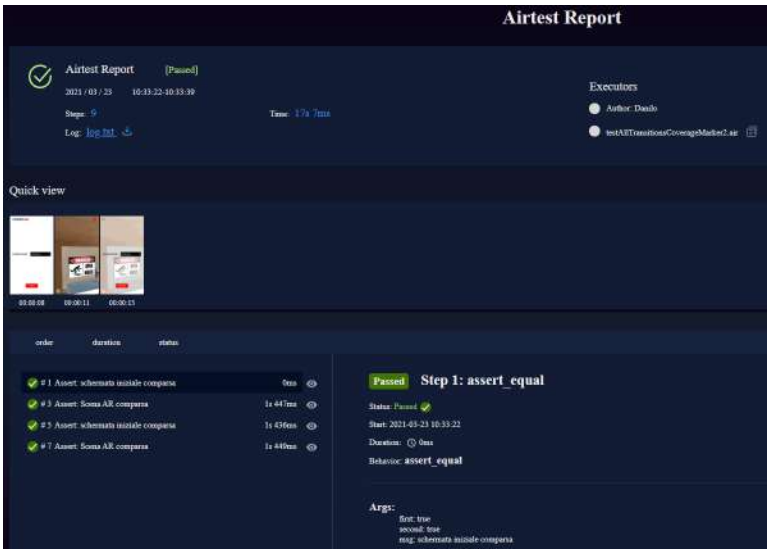


Figura 7.80: Report script test All-Transitions Coverage marker 2

```
Unico paths All Transitions Coverage Marker 2
cliccato bottone: StartButton
cliccato bottone: Settings
cliccato bottone: ResumeButton
Test concluso senza errori
```

Figura 7.81: File interazioni All Transitions Coverage Marker 2

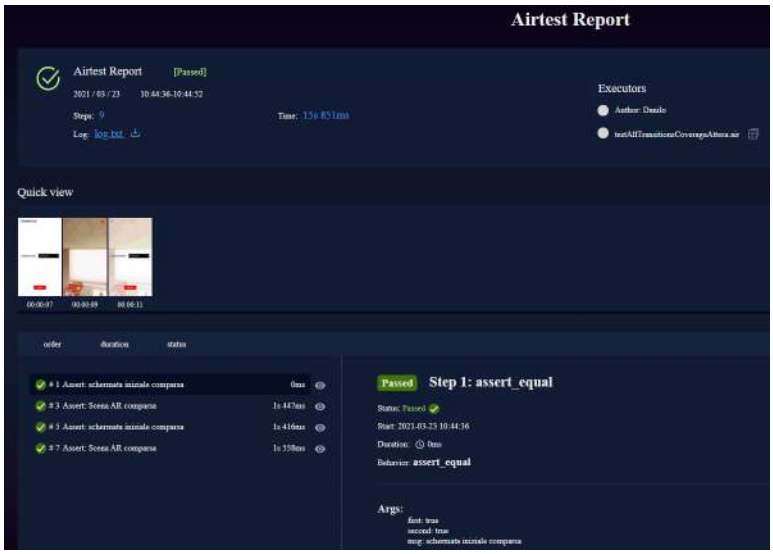


Figura 7.82: Report script test All-Transitions Coverage assenza marker

```
Unico paths All Transitions Coverage Attesa
cliccato bottone: StartButton
cliccato bottone: Settings
cliccato bottone: ResumeButton
Test concluso senza errori
```

Figura 7.83: File interazioni All Transitions Coverage assenza marker

Script test suite All-OneLoop-Paths Coverage

Anche in questo caso è stato necessario implementare tre script di test per gli stessi motivi di cui sopra. Infatti la test suite, teorizzata nella sezione 7.2.3, presenta paths in cui è necessario il riconoscimento del marker 1 o del marker 2 o l'assenza di entrambi. Quindi, si può facilmente intuire, che tale ragione ha comportato la separazione degli script. Le funzioni utilizzate nei tre script sono esattamente uguali ai corrispettivi script della test suite precedente, perciò si mostrerà solo l'ordine con cui sono state richiamate. Si comincia mostrando lo script

relativo all'assenza di entrambe le scene AR, figura 7.84, in cui sono implementati i paths di lunghezza due e tre.

```
 94  f = open("AllOneLoopPathsCoverageInterazioniAttesa.txt","w+");
 95  f.write("Paths All One Loop Paths Coverage Attesa \n");
 96
 97
 98  avviaApp();
 99  verificaSchermataSelezioneLingua("avvio");
100  f.write("\nPATHS LUNGHEZZA 2 \n");
101  f.write("path 1\n");
102  #PATHS DI LUNGHEZZA DUE 4
103  apriMenuTendina();
104  for i in range(len(listaLingue)):
105      if(i == len(listaLingue)-1):
106          scorriMenuTendina();
107      print(listaLingue[i].attr('name'))
108      selezionaLingua(listaLingue[i].attr('name'));
109      verificaSchermataSelezioneLingua("avvio");
110      chiudiApp();
111      avviaApp();
112      verificaSchermataSelezioneLingua("avvio")
113      if(i < len(listaLingue)-1):
114          f.write("\npath "+str(i+2)+"\n");
115      if(i != len(listaLingue)-1):
116          apriMenuTendina();
117
118
119  #PATHS DI LUNGHEZZA TRE 4 + 1
120  f.write("\nPATHS LUNGHEZZA 3 \n");
121  f.write("path 1\n");
122  apriMenuTendina();
123  for i in range(len(listaLingue)):
124      if(i == len(listaLingue)-1):
125          scorriMenuTendina();
126      print(listaLingue[i].attr('name'))
127      selezionaLingua(listaLingue[i].attr('name'));
128      verificaSchermataSelezioneLingua("avvio");
129      bottone.click();
130      f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
131      verificaAssenzaScenaAr();
132      chiudiApp();
133      avviaApp();
134      verificaSchermataSelezioneLingua("avvio")
135      f.write("\npath "+str(i+2)+"\n");
136      if(i != len(listaLingue)-1):
137          apriMenuTendina();
138
139  bottone.click();
140  f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
141  verificaAssenzaScenaAr();
142  bottone.click();
143  f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
144  verificaSchermataSelezioneLingua("appAvviata");
145  chiudiApp();
146  f.write("\nTest concluso senza errori \n");
```

Figura 7.84: Script test All-One-Loop-Paths Coverage assenza marker

In particolare per i paths di lunghezza due ciò che si fa è iterare, per ognuna delle quattro lingue, eseguendo l'avvio dell'applicazione,

verificando la comparsa della schermata iniziale, aprendo il menù a tendina, selezionando la lingua della particolare iterazione, verificando di trovarsi ancora nella schermata iniziale e chiudendo l'applicazione. Per i paths di lunghezza tre si utilizza lo stesso approccio, con l'unica differenza di cliccare il bottone "Start" e verificare che non sia presente nessuna scena AR prima di chiudere l'applicazione. Infine l'ultimo path di lunghezza tre è ottenuto avviando l'applicazione, verificando la presenza della schermata iniziale, cliccando il tasto "Start", verificando l'assenza delle scene AR, cliccando il bottone dell'ingranaggio, verificando la comparsa della schermata di selezione lingua e chiudendo l'applicazione. Passando ai paths di lunghezza quattro, alcuni di essi prevedono il riconoscimento del marker 1 e altri il riconoscimento del marker 2. In figura 7.85 è mostrato lo script che implementa i paths di lunghezza quattro relativi al marker 1. In particolare l'approccio seguito prevede l'iterazione, per ognuna delle lingue selezionabili, effettuando i seguenti step: avvio applicazione, verifica comparsa schermata iniziale, apertura menù a tendina, selezione lingua della particolare iterazione, verifica di permanenza nella schermata iniziale, click del bottone "Start", verifica della comparsa della scena AR relativa al marker 1 e chiusura dell'applicazione. Si noti che dopo il click del tasto "Start" si dovrebbe verificare prima l'assenza della scena AR e successivamente, in seguito al riconoscimento del marker 1, la comparsa della scena AR. Tuttavia ciò non è possibile effettuar-

lo dato che, avendo preventivamente orientato il dispositivo emulato per inquadrare il marker, il suo avvenuto riconoscimento è istantaneo nel momento in cui si avvia la fotocamera. Oltre a questi, vi è un altro path di lunghezza cinque che è possibile ottenere: aprendo l'applicazione, verificando la presenza della schermata iniziale cliccando il bottone "Start", verificando la presenza della scena AR, cliccando il bottone dell'ingranaggio, verificando l'avvenuta comparsa della schermata di selezione lingua e chiudendo l'applicazione. Esattamente gli stessi paths è possibili ottenerli anche con il riconoscimento del marker 2, per questa ragione si evita di riportare lo screen dello script.

```
97  f = open("AllOneLoopPathsCoverageInterazioniMarker1.txt","w+");
98
99  f.write("Paths All One Loop Paths Coverage Marker 1 \n");
100
101 avviaApp();
102 verificaSchermataSelezioneLingua("avvio");
103 f.write("\nPATHS LUNGHEZZA 4 \n");
104 f.write("path 1\n");
105
106 #PATHS DI LUNGHEZZA QUATTRO 4 + 1
107 apriMenuTendina();
108 for i in range(len(listaLingue)):
109     if(i == len(listaLingue)-1):
110         scorriMenuTendina();
111     selezionaLingua(listaLingue[i].attr('name'));
112     verificaSchermataSelezioneLingua("avvio");
113     bottone.click();
114     f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
115     verificaScenaAr();
116     chiudiApp();
117     avviaApp();
118     verificaSchermataSelezioneLingua("avvio")
119     f.write("\npath "+str(i+2)+"\n");
120     if(i != len(listaLingue)-1):
121         apriMenuTendina();
122
123 bottone.click();
124 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
125 verificaScenaAr();
126 bottone.click();
127 f.write("cliccato bottone: "+ bottone.attr('name')+"\n");
128 verificaSchermataSelezioneLingua("appAvviata");
129 chiudiApp();
130 f.write("\nTest concluso senza errori \n");
```

Figura 7.85: Script test All-One-Loop-Paths Coverage marker 1

In definitiva i paths implementati sono risultati essere diciannove, sei in meno di quelli teorizzati nella sezione 7.2.3. I paths non implementati sono i seguenti:

- "Seleziona lingua" - "Attesa" - "Seleziona lingua" - "Sollevamento";
- "Seleziona lingua" - "Attesa" - "Seleziona lingua" - "Caduta";
- "Seleziona lingua" - "Attesa" - "Sollevamento" - "Attesa";
- "Seleziona lingua" - "Attesa" - "Caduta" - "Attesa";
- "Seleziona lingua" - "Attesa" - "Sollevamento" - "Seleziona lingua" - "Caduta";
- "Seleziona lingua" - "Attesa" - "Caduta" - "Seleziona lingua" - "Sollevamento"

Tali paths, come anticipato, non è stato possibile implementarli da script poiché prevedono transizioni di disconoscimento dei marker, di riconoscimento del marker 1 o del marker 2 dopo aver effettuato interazioni nello stato di attesa o di cambio marker. Anche in questo caso, per ognuno degli script, sono presenti istruzioni di scrittura su file. In particolare sono creati tre file delle interazioni, uno per ogni script. Essendo che ogni script contiene più paths, nel relativo file saranno presenti le interazioni appartenenti ad ognuno di essi, cosicché in caso

di errore, le uniche interazioni che dovrebbero essere effettuate risulterebbero quelle relative all'ultimo path. Infine si mostrano i report ottenuti degli script e i relativi file delle interazioni.

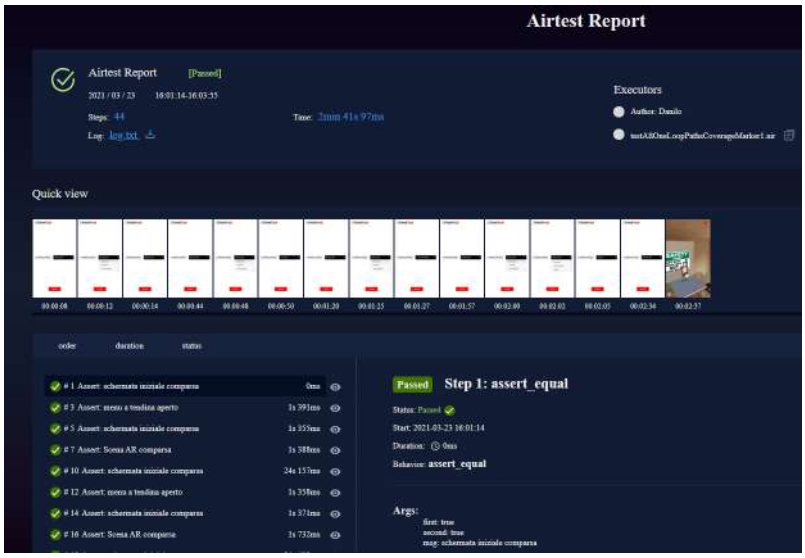


Figura 7.86: Report script test All-One-Loop-Paths Coverage marker 1

```
Paths All One Loop Paths Coverage Marker 1

PATHS LUNGHEZZA 4
path 1
cliccato menu tendina
selezionata lingua: English
cliccato bottone: StartButton

path 2
cliccato menu tendina
selezionata lingua: Italian
cliccato bottone: StartButton

path 3
cliccato menu tendina
selezionata lingua: Lithuanian
cliccato bottone: StartButton

path 4
cliccato menu tendina
selezionata lingua: Urdu
cliccato bottone: StartButton

path 5
cliccato bottone: StartButton
cliccato bottone: Settings

Test concluso senza errori
```

Figura 7.87: File interazioni All One Loop Paths Coverage marker 1

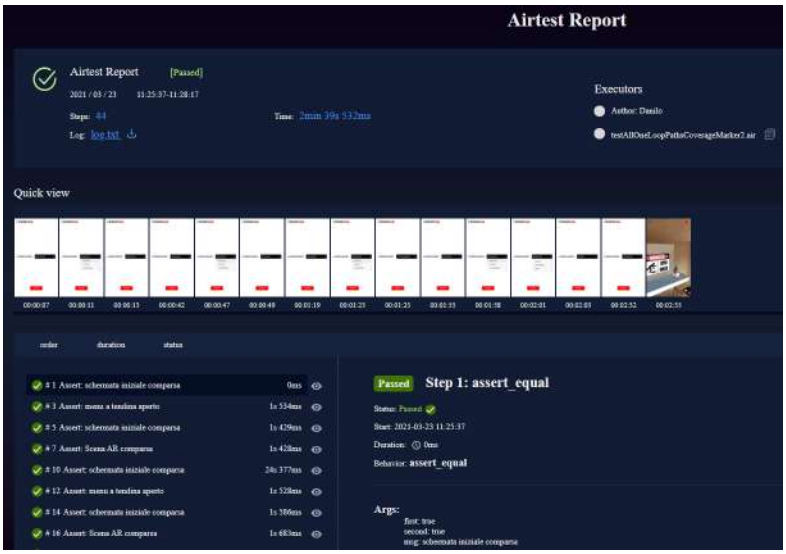


Figura 7.88: Report script test All-One-Loop-Paths Coverage marker 2

```
Paths All One Loop Paths Coverage Marker 2
PATHS LUNGHEZZA 4

path 1
cliccato menu tendina
selezionata lingua: English
cliccato bottone: StartButton

path 2
cliccato menu tendina
selezionata lingua: Italian
cliccato bottone: StartButton

path 3
cliccato menu tendina
selezionata lingua: Lithuanian
cliccato bottone: StartButton

path 4
cliccato menu tendina
selezionata lingua: Urdu
cliccato bottone: StartButton

path 5
cliccato bottone: StartButton
cliccato bottone: Settings

Test concluso senza errori
```

Figura 7.89: File interazioni All One Loop Paths Coverage marker 2

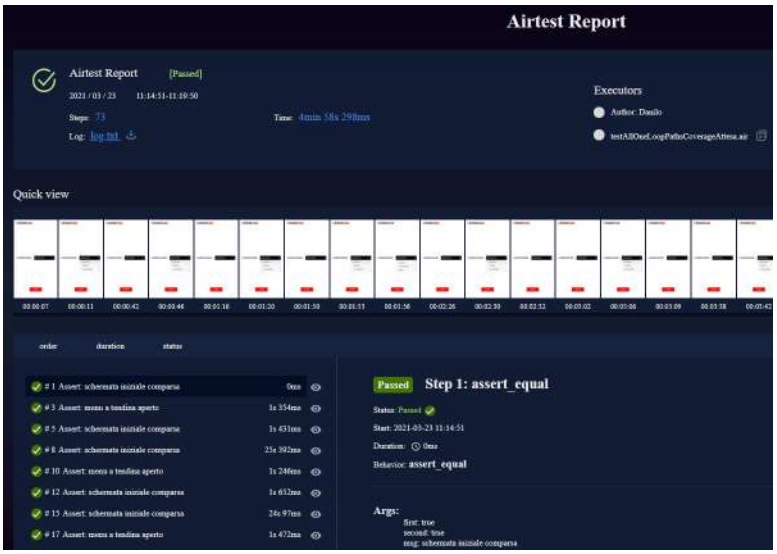


Figura 7.90: Report script test All-One-Loop-Paths Coverage assenza marker

```
Paths All One Loop Paths Coverage Attesa

PATHS LUNGHEZZA 2
path 1
cliccato menu tendina
selezionata lingua: English

path 2
cliccato menu tendina
selezionata lingua: Italian

path 3
cliccato menu tendina
selezionata lingua: Lithuanian

path 4
cliccato menu tendina
selezionata lingua: Urdu

PATHS LUNGHEZZA 3
path 1
cliccato menu tendina
selezionata lingua: English
cliccato bottone: StartButton

path 2
cliccato menu tendina
selezionata lingua: Italian
cliccato bottone: StartButton

path 3
cliccato menu tendina
selezionata lingua: Lithuanian
cliccato bottone: StartButton

path 4
cliccato menu tendina
selezionata lingua: Urdu
cliccato bottone: StartButton

path 5
cliccato bottone: StartButton
cliccato bottone: Settings

Test concluso senza errori
```

Figura 7.91: File interazioni All One Loop Paths Coverage assenza marker

7.2.5 Valutazione copertura delle test suite

Eseguiti gli script di test si valuta quale sia l'effettiva copertura dei rami del codice di ciascuna delle due test suite. Per fare ciò si utilizza lo strumento presentato nella sezione 6.3.2.

Valutazione copertura test suite All-States Coverage

La prima copertura che si valuta è quella relativa alla test suite All-States Coverage. Di seguito si mostra l'analisi effettuata sul log importato dal dispositivo mobile emulato. Si farà riferimento al numero di esecuzioni di ogni ramo, che non è da confondere con il numero di test effettuati. Diversamente dal caso di studio precedente, non sono presenti metodi di tipo "Update", i quali sono eseguiti continuamente dopo che l'applicazione è stata avviata, perciò il numero di esecuzioni sarà confrontato con i paths da eseguire.

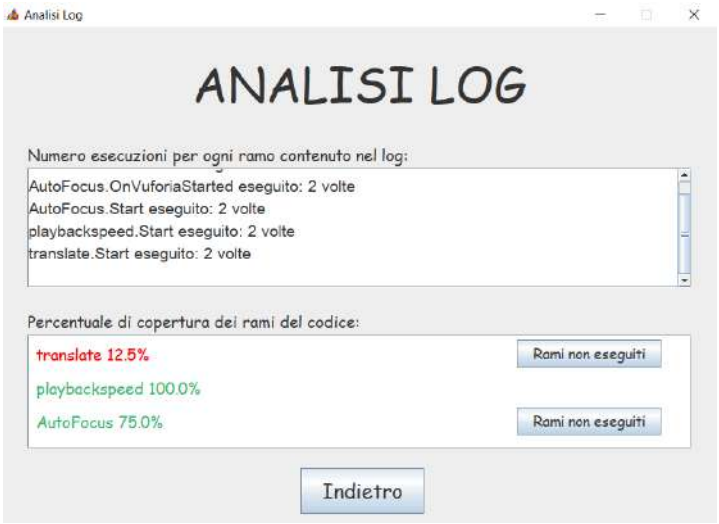


Figura 7.92: Analisi copertura test suite All-States Coverage

Dall'immagine sono visibili tutti i rami presenti nel log e si può notare che sono stati tutti eseguiti due volte, coerentemente con l'esecuzione di ciascuno dei due script di test (sezione 7.2.4) che rappresentano altrettanti paths distinti. Nella tabella successiva si mostrano i rami non presenti all'interno del log.

Rami assenti nel log
AutoFocus.OnPaused.if (!paused)
translate.Destroy
translate.myDropdownValueChangedHandler
translate.myDropdownValueChangedHandler.if (target.value == 0)
translate.myDropdownValueChangedHandler.if (target.value == 1)
translate.myDropdownValueChangedHandler.if (target.value == 2)
translate.myDropdownValueChangedHandler.if (target.value == 3)
translate.SetDropdownIndex

Tabella 7.35: Rami non eseguiti log All-States Coverage

Da questa classificazione dei rami è possibile scaturire la percentuale di copertura di ogni script, presentata anche in figura 7.92. In tabella 7.36 è mostrata la percentuale di ogni script e il relativo rapporto da cui scaturisce. Per rapporto s'intende il numero di rami coperti rispetto al numero di rami totali, riferito a ciascuno degli script.

	AutoFocus	**translate**	**playbackspeed**
Test suite			
All-States	75% ($\frac{3}{4}$)	12.5% ($\frac{1}{8}$)	100% ($\frac{1}{1}$)
Coverage			

Tabella 7.36: Percentuale copertura All-States Coverage

I rami della tabella 7.35 di seguito saranno analizzati nel dettaglio per definire a quali script appartengono e le motivazioni per cui non sono stati eseguiti. Per quanto riguarda la percentuale di copertura dei rami del codice quella più bassa è relativa allo script "translate", atto alla gestione dei cambiamenti di lingua dal menù a tendina. La scarsa copertura di tale script è da ricondurre al fatto che, in questa test suite, non è previsto alcun test case riguardante la selezione delle lingue. Infatti, come si può evincere dalla figura 7.93, tutti i metodi relativi alla “dropdown” (menù a tendina) non sono stati eseguiti, per i motivi precedentemente esposti. Circa i metodi di “Destroy” e “SetDropdownIndex”, le motivazioni della loro mancata esecuzione saranno trattate successivamente nella sezione 7.2.5. Riguardo lo script “playbackspeed” ha una copertura totale e non poteva essere altrimenti dato che, come si può vedere dallo script di figura 7.60, contiene un unico metodo che è eseguito non appena si avvia l’applicazione. In ultimo, lo script “AutoFocus” presenta un unico ramo non eseguito, ugualmente al caso di studio precedente. Esso è relativo al caso in cui l’applicazione fosse messa in pausa e successivamente ripresa.

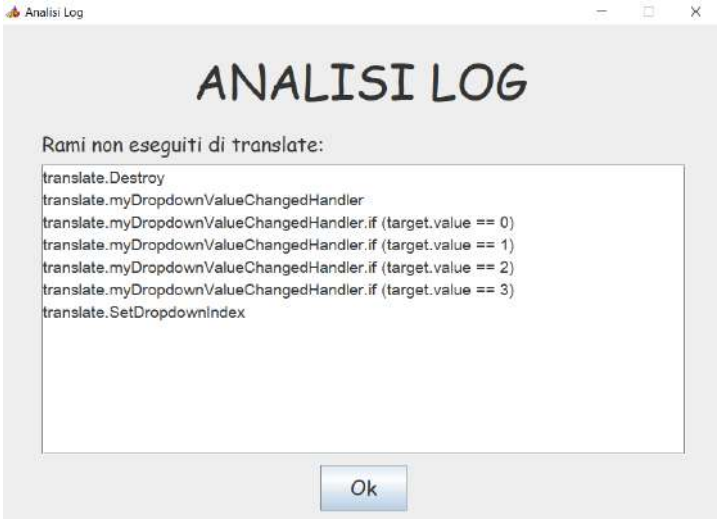


Figura 7.93: Rami non coperti translate All-States Coverage

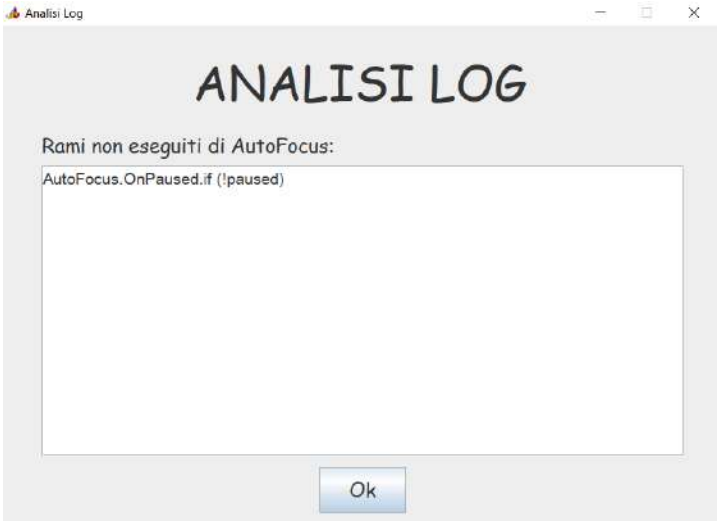


Figura 7.94: Rami non coperti AutoFocus All-States Coverage

Valutazione copertura test suite All-Transitions Coverage

Ultimata l'analisi della test suite relativa ad All-State Coverage, si passa ora a quella scaturita dal criterio All-Transitions Coverage. Di seguito si riporta l'analisi ad essa relativa.

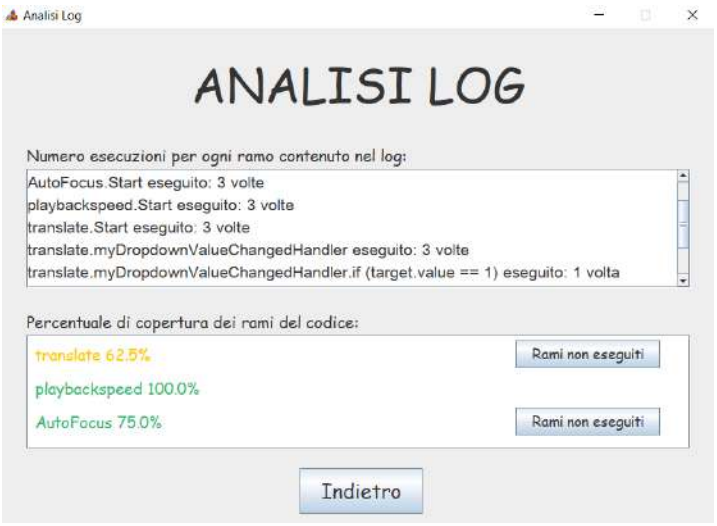


Figura 7.95: Analisi copertura test suite All-Transitions Coverage

Dato che dall'immagine sono visibili solo alcuni dei rami presenti nel log, si riporta la tabella 7.37 che li mostra tutti.

Rami presenti nel log
AutoFocus.OnPaused
AutoFocus.OnVuforiaStarted
AutoFocus.Start
playbackspeed.Start
translate.Start
translate.myDropdownValueChangedHandler
translate.myDropdownValueChangedHandler.if (target.value == 1)
translate.myDropdownValueChangedHandler.if (target.value == 2)
translate.myDropdownValueChangedHandler.if (target.value == 3)

Tabella 7.37: Rami eseguiti log All-Transitions Coverage

Come si può notare dalla tabella 7.37, gli script che presentano il

metodo "Start" sono stati eseguiti tre volte, coerentemente con l'esecuzione di ciascuno dei tre script di test (sezione 7.2.4) che rappresentano altrettanti paths distinti. Come si può intuire dalla tabella 7.37, la copertura sarà migliore del caso precedente poiché vi è un maggiore presenza di rami nel log. Nella tabella successiva si mostrano i rami non presenti all'interno del log.

Rami assenti nel log
AutoFocus.OnPaused.if (!paused)
translate.Destroy
translate.myDropdownValueChangedHandler.if (target.value == 0)
translate.SetDropdownIndex

Tabella 7.38: Rami non eseguiti log All-Transitions Coverage

Da questa classificazione dei rami è possibile scaturire la percentuale di copertura di ogni script, presentata anche in figura 7.95. In tabella 7.39 è mostrata la percentuale di ogni script e il relativo rapporto da cui scaturisce. Per rapporto s'intende il numero di rami coperti rispetto al numero di rami totali, riferito a ciascuno degli script.

	AutoFocus	**AnimalSpawn**	**AnimalsController**
Test suite All-States Coverage	75% ($\frac{3}{4}$)	62.5% ($\frac{5}{8}$)	100% ($\frac{1}{1}$)

Tabella 7.39: Percentuale copertura All-Transitions Coverage

I rami della tabella 7.38 di seguito saranno analizzati nel dettaglio per definire a quali script appartengono e le motivazioni per cui non sono stati eseguiti. Riguardo la copertura dello script "translate", risulta essere nettamente migliore rispetto al caso precedente perché, nella corrente test suite, è prevista la copertura delle transizioni riguardanti la selezione della lingua. Come si può vedere dall'immagine 7.96, i rami non eseguiti riguardano:

- Il caso in cui si selezioni la lingua avente indice 0 (inglese). Tale ramo risulta non essere coperto, nonostante vi fosse una transizione di scelta di tale lingua, poichè l'inglese è impostato come lingua di default e di conseguenza aprire il menù a tendina e riselezionare la lingua inglese, non sortisce l'attivazione del metodo. Ciò è dovuto al fatto che tale metodo è attivato solo nel momento in cui si seleziona una lingua diversa da quella attualmente selezionata;

- I metodi "Destroy" e "SetDropdownIndex", che come anticipato saranno trattati nella sezione 7.2.5.

Riguardo la copertura dei rimanenti script è rimasta invariata e si rimanda alla precedente sezione per le motivazioni.

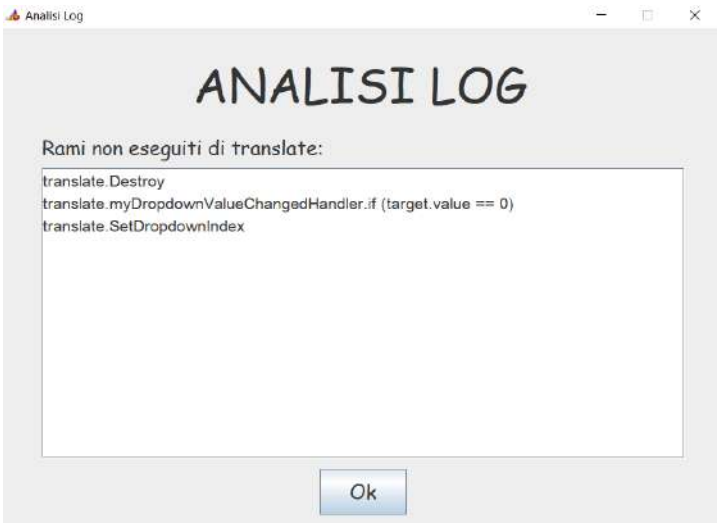


Figura 7.96: Rami non coperti translate All-Transitions Coverage

Valutazione copertura test suite All-One-Loop-Paths Coverage

In ultimo si presenta l'analisi della test suite scaturita dal criterio All-One-Loop-Paths Coverage. Di seguito si riporta l'analisi ad essa relativa.

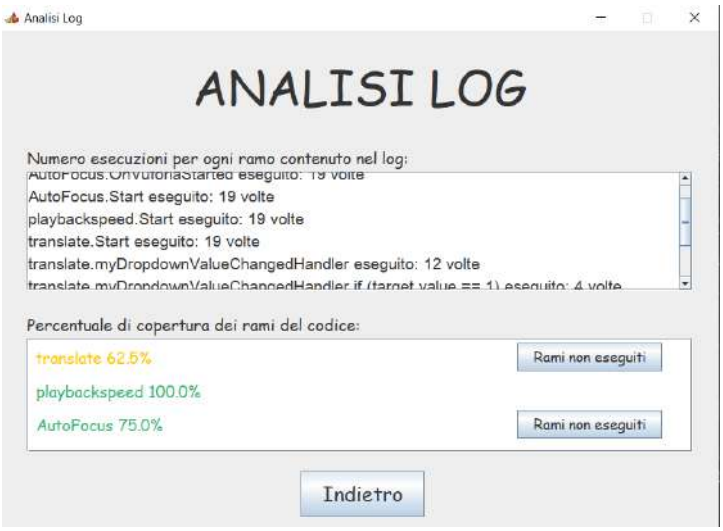


Figura 7.97: Analisi copertura test suite All-One-Loop-Paths Coverage

Dato che dall'immagine sono presenti solo alcuni dei rami presenti nel log, in tabella 7.40 se ne mostra l'intera lista. Come si può notare dalla figura gli script che presentano il metodo "Start" sono stati eseguiti diciannove volte, coerentemente con la somma del numero di paths dei tre script di test (sezione 7.2.4). Confrontando la tabella 7.40 con la 7.37, si può notare che presentano esattamente gli stessi rami coperti e, di conseguenza, si può intuire che abbiano la stessa copertura (come mostrato in figura 7.97).

Rami presenti nel log
AutoFocus.OnPaused
AutoFocus.OnVuforiaStarted
AutoFocus.Start
playbackspeed.Start
translate.Start
translate.myDropdownValueChangedHandler
translate.myDropdownValueChangedHandler.if (target.value == 1)
translate.myDropdownValueChangedHandler.if (target.value == 2)
translate.myDropdownValueChangedHandler.if (target.value == 3)

Tabella 7.40: Rami eseguiti log All-One-Loop-Paths Coverage

Dualmente, come mostrato in tabella 7.41, si avranno anche gli stessi rami non coperti del caso precedente e per questa ragione si evita spiegarne le motivazioni e di riportarne gli screen.

Rami assenti nel log
AutoFocus.OnPaused.if (!paused)
translate.Destroy
translate.myDropdownValueChangedHandler.if (target.value == 0)
translate.SetDropdownIndex

Tabella 7.41: Rami non eseguiti log All-One-Loop-Paths Coverage

	AutoFocus	**AnimalSpawn**	**AnimalsController**
Test suite All-States Coverage	75% ($\frac{3}{4}$)	62.5% ($\frac{5}{8}$)	100% ($\frac{1}{1}$)

Tabella 7.42: Percentuale copertura All-One-Loop-Paths Coverage

Confronto tra le test suite

Dai risultati ottenuti nelle precedenti sezioni, si confrontano le test suite per definire quale fra esse risulta essere migliore per il testing dell'applicazione in esame. E' lampante che le test suite scaturite dal criterio All-Transitions Coverage e All-One-Loop-Pathts Coverage risultino essere migliori di quella scaturita dal criterio All-States Coverage, poiché presentano una copertura dei rami di codice nettamente superiore, come mostrato nella tabella 7.43.

	AutoFocus	**translate**	**playbackspeed**
Test suite All-States Coverage	75% ($\frac{3}{4}$)	12.5% ($\frac{1}{8}$)	100% ($\frac{1}{1}$)
Test suite All-Transitions Coverage	75% ($\frac{3}{4}$)	62.5% ($\frac{5}{8}$)	100% ($\frac{1}{1}$)
Test suite All-One-Loop-Paths Coverage	75% ($\frac{3}{4}$)	62.5% ($\frac{5}{8}$)	100% ($\frac{1}{1}$)

Tabella 7.43: Confronto percentuale copertura

	N° test case
Test suite All-States Coverage	4
Test suite All-Transitions Coverage	15
Test suite All-One-Loop-Paths Coverage	64

Tabella 7.44: Confronto numero test case

Definito ciò, si valutano le motivazioni per cui le funzioni "Destroy" e "SetDropdownIndex" non siano state mai invocate in nessuna delle test suite eseguite. Analizzando il progetto in Unity si è notato che ta-

li funzioni non hanno alcun legame con l'esecuzione dell'applicazione. Riguardo la funzione "Destroy" è atta a deallocare i listener associati al menù a tendina. Tuttavia in "PointAR" non vi è alcun bottone che permetta l'uscita "controllata" dall'applicazione e, di conseguenza, tale funzione non è richiamata da alcun oggetto. Relativamente alla funzione "SetDropdownIndex", dovrebbe avere la funzionalità di catturare l'indice della lingua selezionata ma anch'essa non risulta essere richiamata da alcun oggetto. Per provare quanto detto, tale funzione è stata cancellata e l'applicazione così modificata è stata deployata sul dispositivo emulato. Come previsto, il funzionamento di "PointAR" non ha subito alcun cambiamento. In definitiva si può concludere che le funzioni "Destroy" e "SetDropdownIndex" sono un esempio di "dead code", ossia istruzioni che non saranno mai eseguite. Si passa alla valutazione della test suite migliore ed essendo che la test suite All-Transitions Coverage e All-One-Loop-Pathts Coverage hanno la stessa copertura, per definire quale tra esse sia la migliore, si valuta l'efficacia attraverso il numero di test case presenti in ognuna delle test suite. Com'è possibile notare dal confronto tra le righe della tabella 7.44, il numero di test case relativo ad All-Transition è decisamente inferiore rispetto a quello associato ad All-One-Loop-Paths e, pertanto, si ritiene la test suite All-Transitions Coverage migliore. Definita tale test suite come la migliore tra quelle proposte, è lecito domandarsi se la copertura di tale test suite possa essere migliorata ancor di più. Per rispondere a

questo quesito bisogna analizzare i rami degli script che non risultano avere una copertura totale. Cominciando dallo script "AutoFocus", l'unico ramo che non risulta essere coperto è quello relativo al caso in cui l'applicazione sia messa in pausa e successivamente ripresa. Questa funzionalità è relativa alla categoria di eventi di sistema Android e, come tale, esulava dagli oneri del modello proposto in figura 7.66. Aggiungendo tale test case alla test suite All-Transitions Coverage si raggiunge una copertura completa per lo script in questione. Le righe di codice aggiunte in coda allo script sono le stesse di quelle mostrate nel precedente caso di studio, figura 7.33. Per quanto riguarda il ramo non coperto relativo allo script "translate", è relativo alla selezione della lingua inglese in sostituzione di una precedentemente selezionata. Tale test prevede che lo stato "Seleziona lingua" sia ripetuto tre volte e in nessuna test suite teorizzata vi è questa ripetizione. Tale caso sarebbe stato coperto se si fosse teorizzato il criterio All-Two-Loop-Paths Coverage, che, rispetto all'All-One-LoopPaths Coverage, prevede paths contenenti la tripla ripetizione di uno stato. Tuttavia tale criterio avrebbe previsto un numero di paths maggiore rispetto all'All-One-LoopPaths Coverage, con conseguente aumento del numero di test case. Quindi la scelta che si è fatta è stata semplicemente quella di aggiungere il path in grado di eseguire il ramo scoperto. Il codice che implementa quanto detto è mostrato in figura 7.98 e può essere aggiunto in coda uno qualsiasi dei tre script. Dopo le modifi-

che apportate agli script di test All-Transitions Coverage, si effettua nuovamente l'analisi ottenendo la copertura mostra in figura 7.99.

```
avviaApp();
verificaSchermataSelezioneLingua("avvio");
apriMenuTendina();
for i in range(1,-1,-1):
    if(i == len(listaLingue)-1):
        scorriMenuTendina();
    selezionaLingua(listaLingue[i].attr('name'));
    verificaSchermataSelezioneLingua("avvio");
    if(i != 0):
        apriMenuTendina();

keyevent("KEYCODE_APP_SWITCH")
keyevent("KEYCODE_APP_SWITCH")
chiudiApp()
```

Figura 7.98: Codice per copertura rami

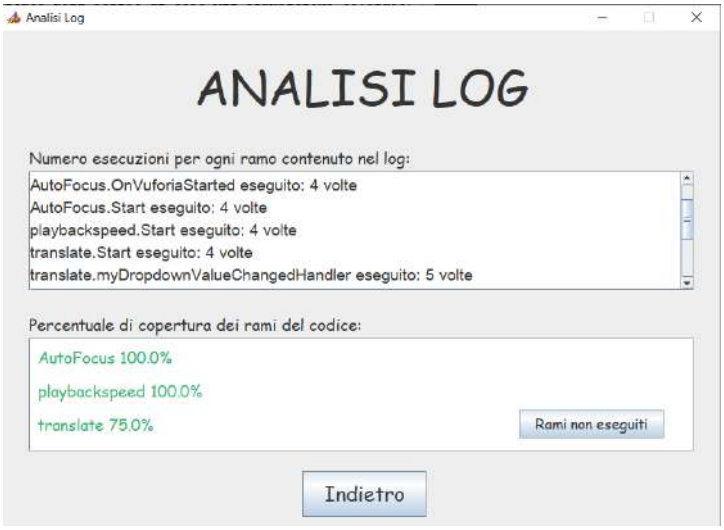


Figura 7.99: Nuova analisi copertura All-Transitions Coverage

In definitiva la migliore copertura possibile che si è riusciti ad ottenere è quella mostrata nella tabella 7.45.

	AutoFocus	**translate**	**playbackspeed**
Test suite All-Transitions Coverage estesa	100% $(\frac{4}{4})$	75% $(\frac{6}{8})$	100% $(\frac{1}{1})$

Tabella 7.45: Nuova percentuale copertura

7.2.6 Analisi Mutazionale

Al fine di valutare la qualità delle test suite ottenute si applicherà l'analisi mutazionale su "PointAR". Tale analisi sarà effettuata su tutte le test suite e non solo su quella ritenuta migliore, ossia quella scaturita dal criterio All-Transitions Coverage, perché è necessario confrontare la capacità di ognuna di scoprire malfunzionamenti. In altre parole potrebbe essere possibile che l'elevata, o mediocre, copertura di codice non sia correlata alla capacità di scoperta dei malfunzionamenti. Quindi il primo passo necessario per effettuare l'analisi mutazionale consta nella definizione del fault model. Analizzando il progetto dell'applicazione unitamente con le action presentate in tabella 7.25, si è pensato di iniettare i seguenti difetti:

1. Impossibilità di apertura del menù a tendina;
2. Errore nella traduzione del cartello, relativo al marker 1, quando si seleziona la lingua lituana.

Gli operatori assegnati sono rispettivamente: deselezione checkbox di GameObject e sostituzione di valori booleani. Come si può intuire, ciò genera due mutanti:

1. Mutante 1: impossibilità di aprire il menù a tendina quando si presenta la schermata per la selezione della lingua, sia all'avvio che dopo il click del bottone dell'ingranaggio. In altre parole non

si sta dando la possibilità all'utente di scegliere la lingua della traduzione, ma lo si sta obbligando a visualizzare gli eventuali cartelli inquadrati in inglese. L'iniezione di tale malfunzionamento si è ottenuta deselezionando la checkbox relativo al GameObject "Dropdown", responsabile del menù a tendina, figura 7.100;

2. Mutante 2: errore nella traduzione del cartello, relativo al marker 1, quando si seleziona la lingua lituana. In altri termini quando l'utente seleziona la lingua lituana, se si inquadra il marker 1 esso presenterà la traduzione in lingua italiana anziché quella lituana. Tale malfunzionamento è stato ottenuto modificando lo script "translate", invertendo il valore booleano dell'istruzione alla riga 71 con quello alla riga 72, come si può notare confrontandolo con quello mostrato in figura 7.62. La porzione di script modificata è presentata in figura 7.101.

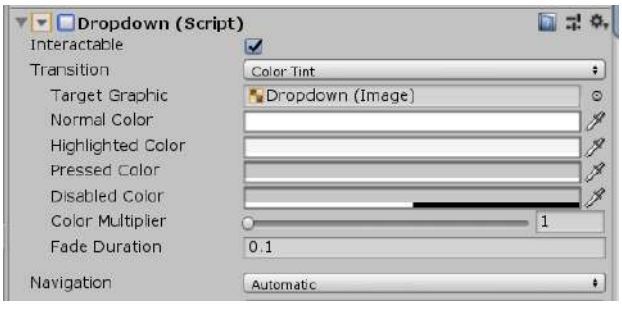


Figura 7.100: Checkbox deselezionata Dropodown

```
65
66          // If "Lithuanian" is selected
67          if (target.value == 2)
68          {
69              SondaManager.inserisciSonda("translate.myDropdownValueChangedHandler.if (target.value == 2)");
70              // Hide non-Lithuanian signage
71              LIFTItalian.SetActive(true);
72              LIFTLithuanian.SetActive(false);
73              LIFTUrdu.SetActive(false);
74              FALLItalian.SetActive(false);
75              FALLLithuanian.SetActive(true);
76              FALLUrdu.SetActive(false);
77          }
```

Figura 7.101: Porzione di script translate modificato

In ultimo è fondamentale presentare gli oracoli al fine di valutare se le test suite in esame sono in grado di uccidere i mutanti. Gli oracoli che saranno descritti, sono quelli già introdotti nell'implementazione degli script di test, sezione 7.2.4. Di seguito si mostrano gli oracoli per ogni versione dei mutanti:

1. Oracolo 1: è associato al mutante 1 e consiste nel valutare se il menù a tendina è in grado di mostrare tutte le lingue in esso presenti. In altre parole se l'utente clicca sul menù a tendina, esso deve aprirsi e mostrare le diverse lingue selezionabili. A livello pratico ciò sarà verificato mediante l'assert associato a tale interazione, contenuto nella funzione "apriMenuTendina";

2. Oracolo 2: è associato al mutante 2 e consiste nel verificare la corretta traduzione del cartello (marker 1) inquadrato. In altre parole è necessario che il cartello presenti la traduzione nella lingua selezionata dall'utente. A livello pratico ciò sarà verificato mediante l'assert di corretta comparsa dei cartelli tradotti, contenuto nella funzione "verificaScenaAR".

Dopo aver descritto i mutanti, si passa alla valutazione della qualità delle test suite esposte nelle precedenti sezioni.

Test suite All-States Coverage

Con tale test suite, descritta nella sezione 7.2.3, si valuta quale sia la sua capacità di scoprire i malfunzionamenti, relativi all'apertura del menù a tendina, indotti nel primo mutante. Tale test suite non sarà in grado di individuarlo dato che non prevede nessuna interazione con il menù a tendina. Di seguito si mostrano i report che confermano quanto detto.

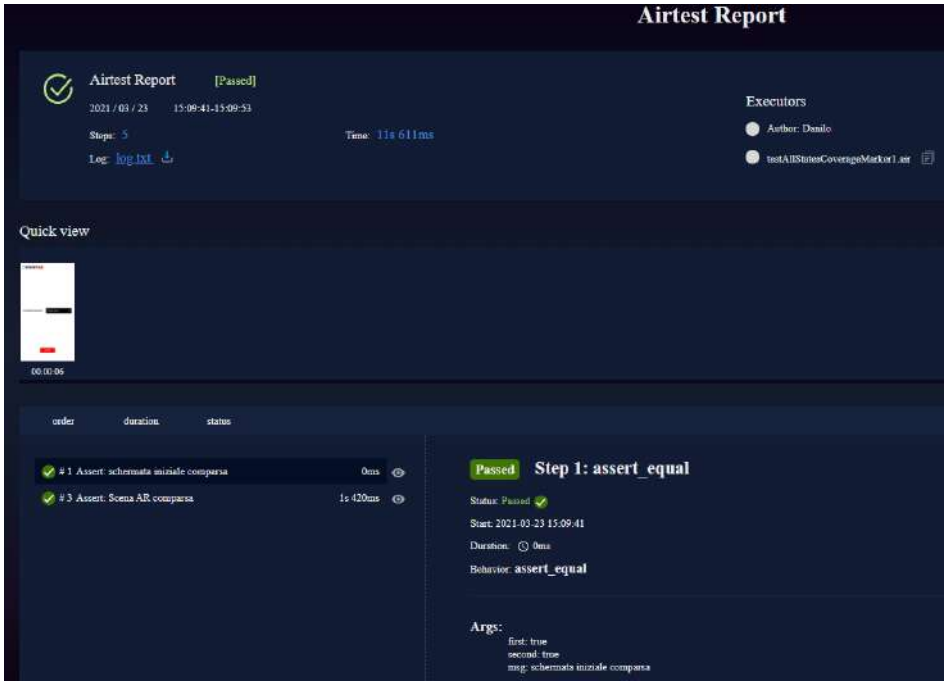


Figura 7.102: Esito script test All-States Coverage marker 1 mutante 1

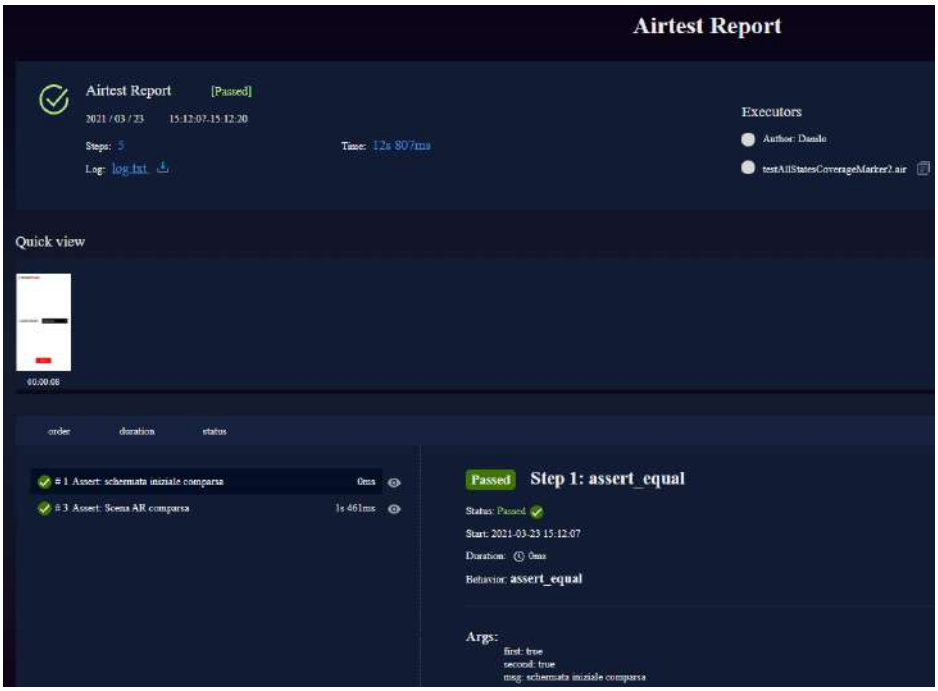


Figura 7.103: Esito script test All-States Coverage marker 2 mutante 1

Come anticipato gli esiti dei report 7.102, 7.103 e quelli degli oracoli, rispettivamente 7.71 e 7.73, risultano essere esattamente gli stessi. I file delle interazioni si è evitato di riportarli poiché sarebbero uguali a quelli di figura 7.72 e 7.74. Per quanto riguarda il secondo mutante dell'applicazione, che prevede il malfunzionamento legato all'errata traduzione del marker 1, anch'esso non sarà mai scoperto dato che non interagendo con il menù a tendina la lingua selezionata sarà sempre quella di default. Di seguito si mostra il report dello script, relativo al marker 1. Anche in questo caso l'esito è lo stesso di quello dell'oracolo di figura 7.71 e si è evitato di riportare i file delle interazioni poiché sarebbero uguali a quelli di figura 7.72 e 7.74.

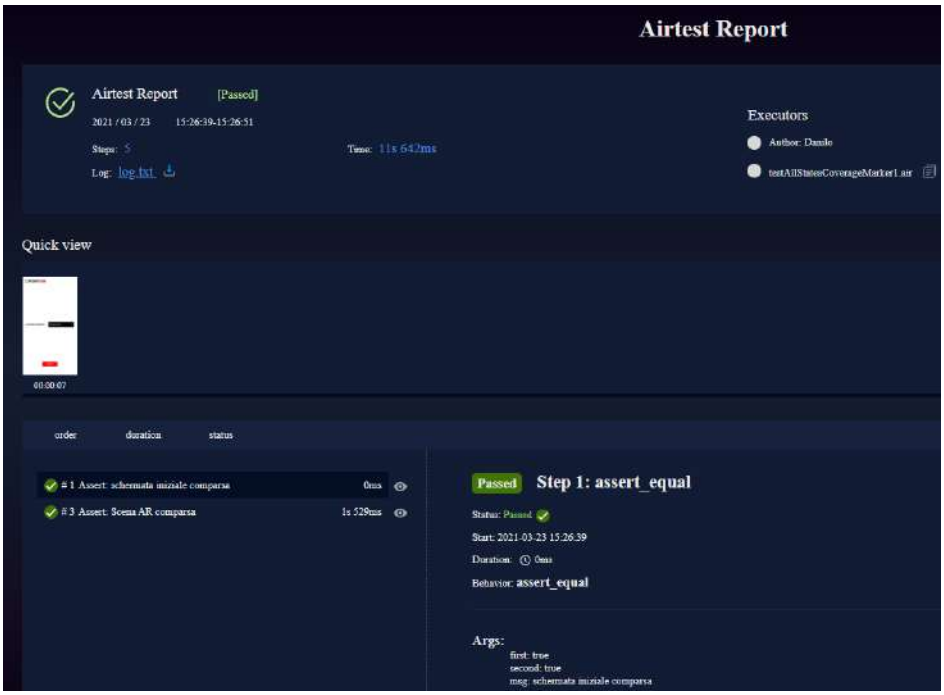


Figura 7.104: Esito script test All-States Coverage marker 1 mutante 2

Test suite All-Transitions Coverage

Con tale test suite, descritta nella sezione 7.2.3, si valuta quale sia la sua capacità di scoprire i malfunzionamenti, relativi all'apertura del menù a tendina, indotti nel primo mutante. Tale test suite, diversamente dalla precedente, sarà sicuramente in grado di rilevare l'errore poiché effettuerà tutte le transizioni possibili, e, di conseguenza, indubbiamente effettuerà quelle relative alla selezione della lingua. Di seguito si mostra il report e il file delle interazioni scaturenti dallo script di test applicato questo mutante dell'applicazione.

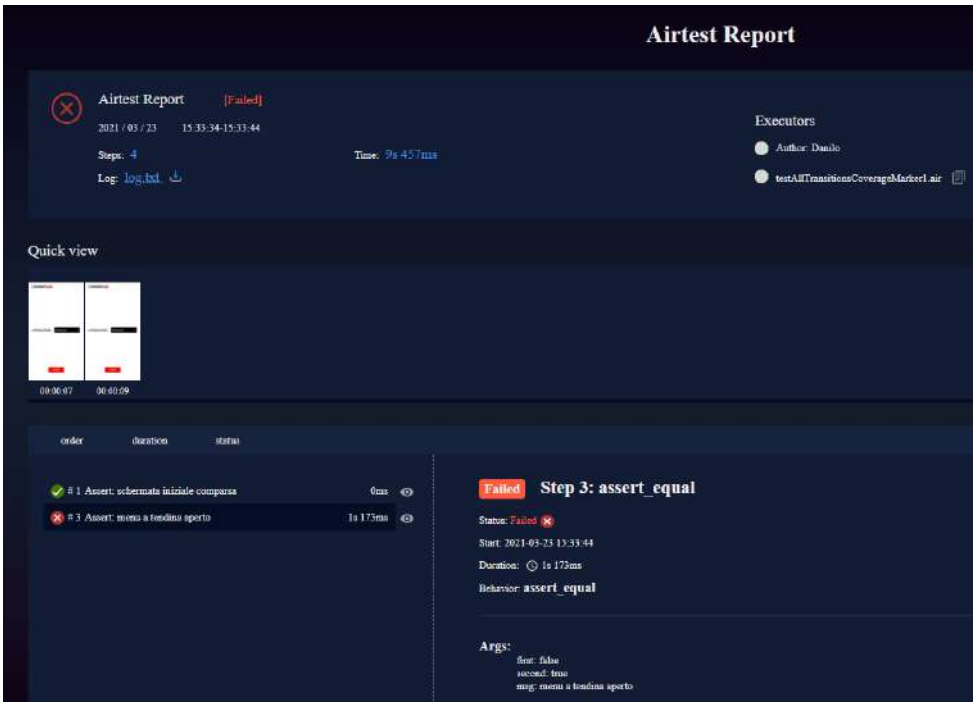


Figura 7.105: Esito script test All-Transitions Coverage marker 1 mutante 1

```
Unico paths All Transitions Coverage Marker 1
cliccato menu tendina
```

Figura 7.106: File interazioni All-Transitions Coverage marker 1 mutante 1

Come si può notare, il report mostra il malfunzionamento rilevato rispetto alla non apertura del menù a tendina. Infatti confrontandolo con l'oracolo di figura 7.107, si può notare che l'assert relativo all'apertura del menù a tendina risulta essere fallito, così come il file delle interazioni 7.106 rispetto a quello dell'oracolo (7.79) non presenta la stringa finale di conclusione con successo del test. Si noti che non sono stati eseguiti gli altri script di test, relativi alla medesima test suite, poiché, per dimostrare l'uccisione del mutante, è sufficiente che vi sia almeno un test case che abbia esito diverso da quello dell'oracolo.

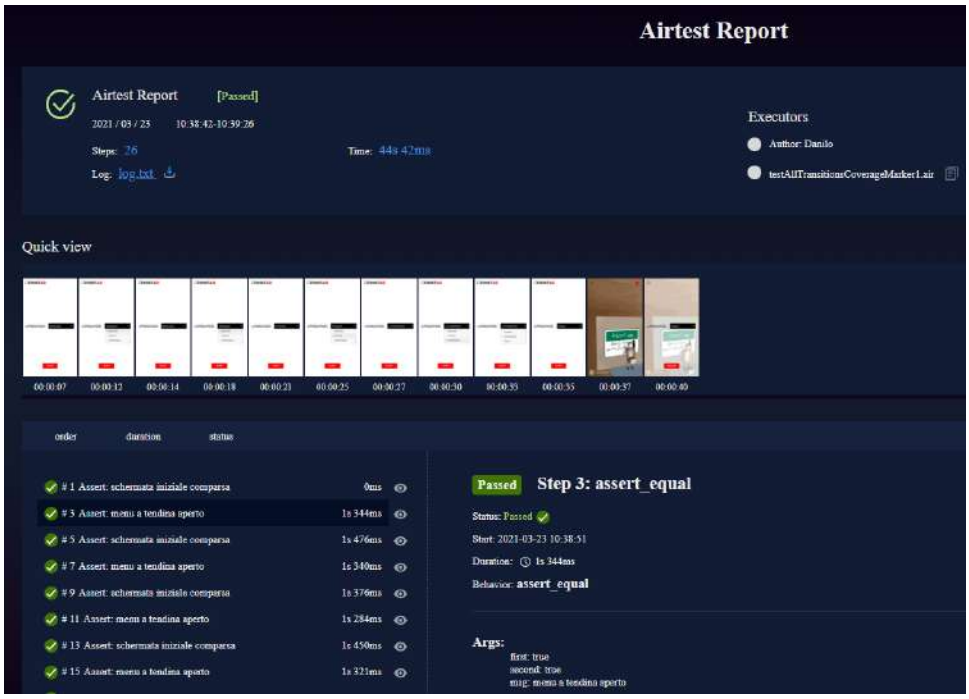


Figura 7.107: Oracolo All-Transitions Coverage marker 1 mutante 1

Per quanto riguarda il secondo mutante di "PointAR", la test suite in esame non sarà in grado di rilevarne il malfunzionamento. Ciò è dovuto al fatto che per ogni lingua non si procede a valutare l'effettiva corretta traduzione del marker. In particolare nella test suite della sezione 7.2.3, si sono effettuate tutte le transizioni relative alla selezione della lingua e solo con l'ultima di esse (urdu), si valuta la correttezza della scena di realtà aumentata. A riprova di quanto detto, in figura 7.108, si mostra il report dello script, relativo al marker 1. In questo caso l'esito è lo stesso di quello dell'oracolo di figura 7.78. Si è evitato di riportare il file delle interazioni poiché sarebbe uguale a quello di figura 7.72.

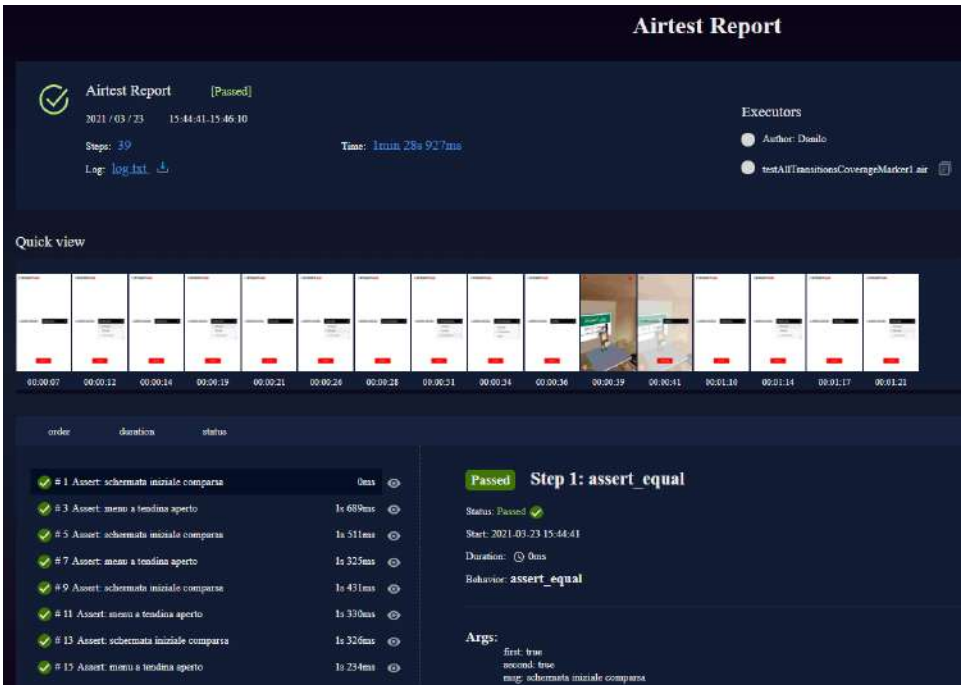


Figura 7.108: Esito script test All-Transitions Coverage marker 1 mutante 2

Test suite All-One-Loop-Pathts Coverage

Con tale test suite, descritta nella sezione 7.2.3, si valuta quale sia la sua capacità di scoprire i malfunzionamenti, relativi all'apertura del menù a tendina, indotti nel primo mutante. Così come la precedente, essa sarà in grado di rilevare l'errore, dato che in diversi path prevede l'interazione con il menù a tendina che provoca il malfunzionamento. Di seguito si mostra il report e il file delle interazioni scaturenti dallo script di test applicato al suddetto mutante dell'applicazione.

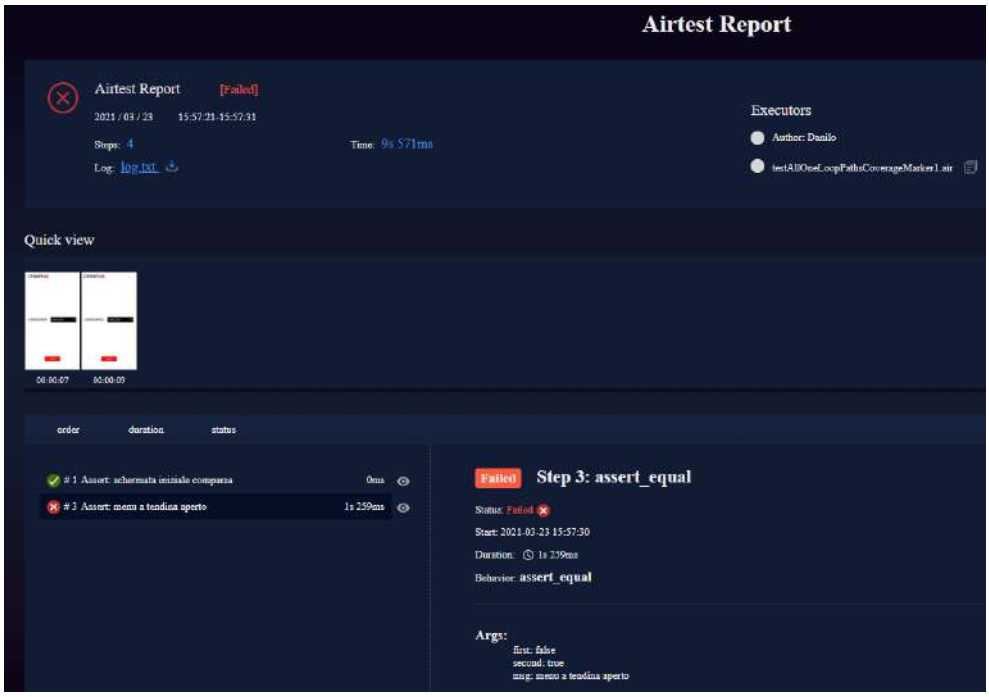


Figura 7.109: Esito script test All-One-Loop-Paths Coverage marker 1 mutante 1

```
Paths All One Loop Paths Coverage Marker 1

PATHS LUNGHEZZA 4
path 1
cliccato menu tendina
```

Figura 7.110: File interazioni All-One-Loop-Paths Coverage marker 1 mutante 1

Come si può notare, il report mostra il malfunzionamento rilevato rispetto alla non apertura del menù a tendina. Infatti confrontandolo con l'oracolo di figura 7.111, si può notare che l'assert relativo all'apertura del menù a tendina risulta essere fallito, così come il file delle interazioni 7.110 rispetto a quello dell'oracolo (7.87) non presenta la stringa finale di conclusione con successo del test.

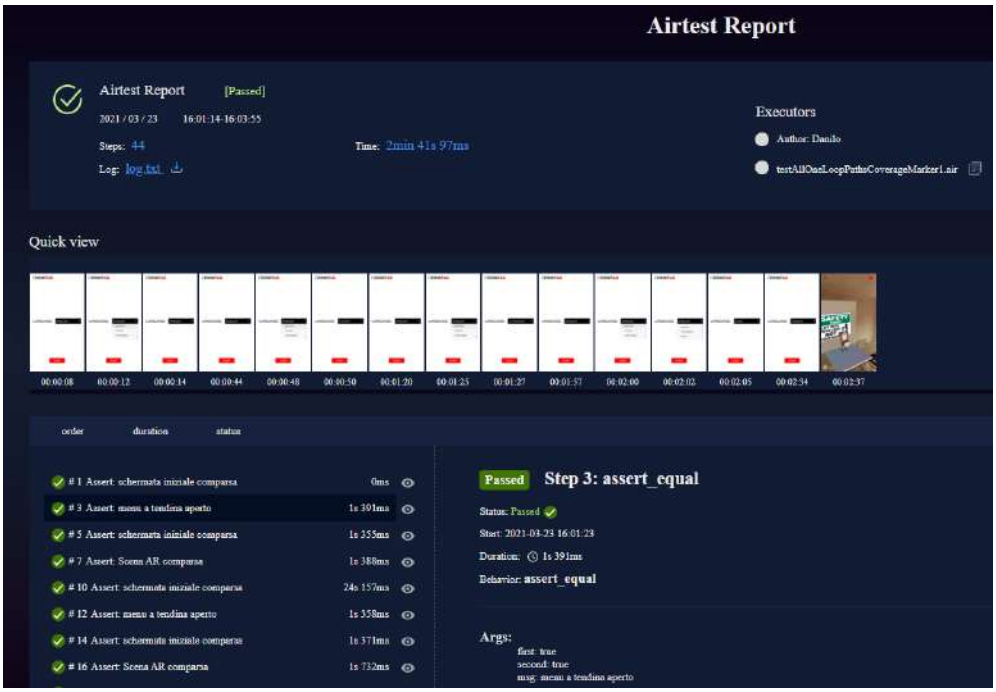


Figura 7.111: Oracolo All-One-Loop-Paths Coverage marker 1 mutante 1

Si noti che non sono stati eseguiti gli altri script di test, relativi alla medesima test suite, poiché, per dimostrare l'uccisione del mutante, è sufficiente che vi sia almeno un test case che abbia esito diverso da quello dell'oracolo. Passando al secondo mutante dell'applicazione tale test suite, diversamente dalle precedenti, sarà in grado di rilevarlo. Ciò è dovuto al fatto che, per ogni lingua selezionabile, è previsto un path che verifichi la corretta comparsa del cartello tradotto. Di seguito sono mostrati il report, il file delle interazioni e l'oracolo.

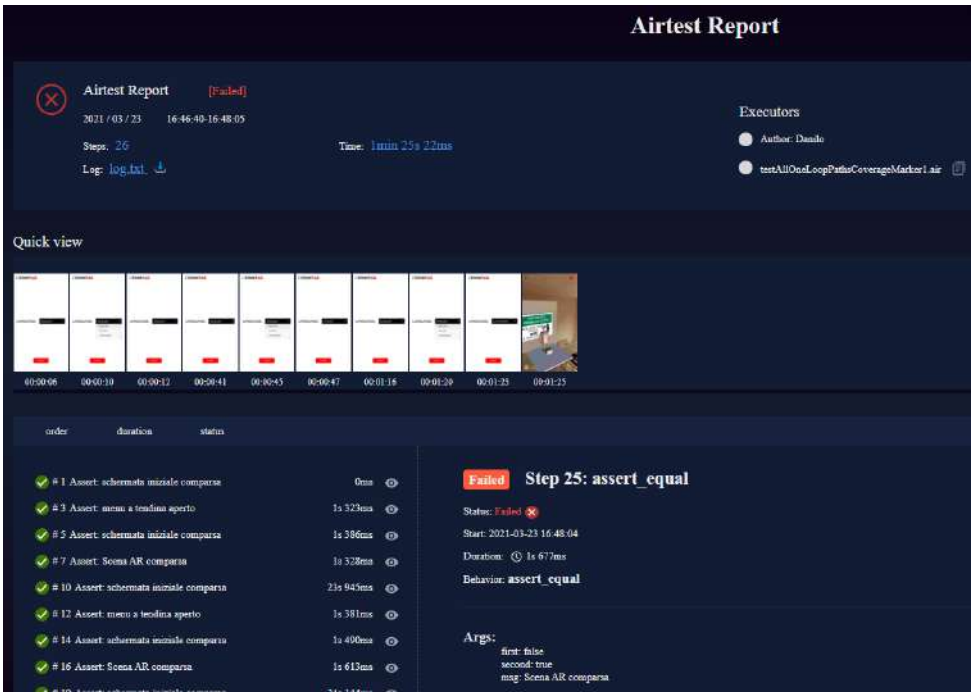


Figura 7.112: Esito script test All-One-Loop-Paths Coverage marker 1 mutante 1

```
Paths All One Loop Paths Coverage Marker 1

PATHS LUNGHEZZA 4
path 1
cliccato menu tendina
selezionata lingua: English
cliccato bottone: StartButton

path 2
cliccato menu tendina
selezionata lingua: Italian
cliccato bottone: StartButton

path 3
cliccato menu tendina
selezionata lingua: Lithuanian
cliccato bottone: StartButton
```

Figura 7.113: File interazioni All-One-Loop-Paths Coverage marker 1 mutante 1

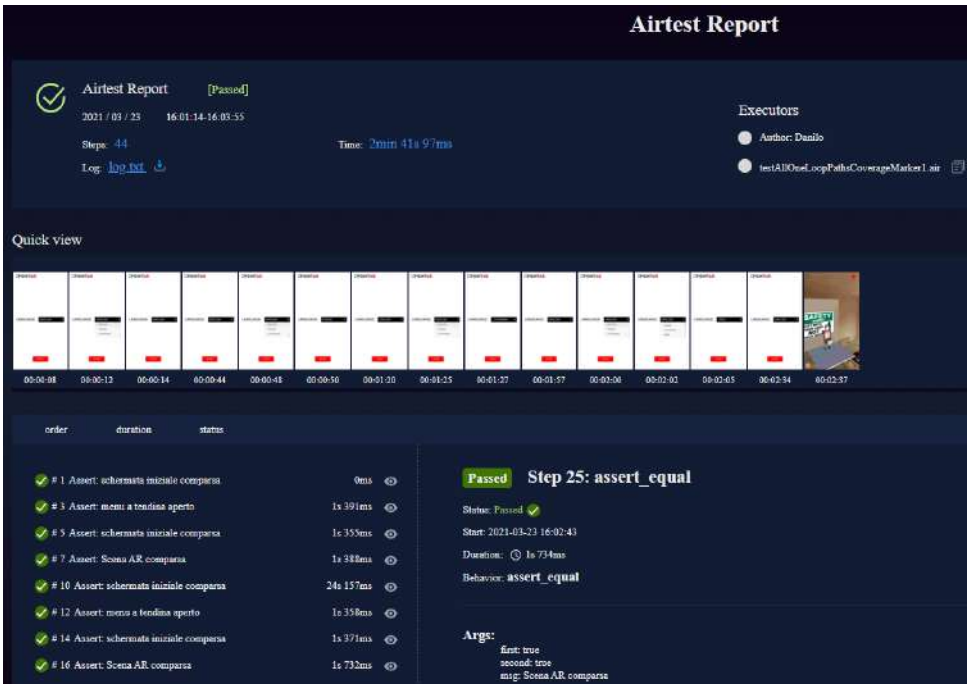


Figura 7.114: Oracolo All-One-Loop-Paths Coverage marker 1 mutante 1

Come si può notare, il malfunzionamento risulta essere rilevato poiché l'assert, relativo alla corretta comparsa della scena AR, risulta essere fallito, così come il file delle interazioni 7.113 rispetto a quello dell'oracolo (7.87) non presenta la stringa finale di conclusione con successo del test.

Confronto

Come si è potuto intuire, la test suite scaturita dal criterio All-One-Loop-Pathts Coverage risulta essere migliore delle altre. Tuttavia, per dimostrarlo matematicamente, si effettua il calcolo dell'efficacia di ognuna delle test suite, utilizzando la formula mostrata nella sezione 6.3.3. Riguardo la test suite di All-States Coverage, essa non è in

grado di uccidere alcun mutante per tale ragione il calcolo dell'efficacia è effettuato come segue:

$$TER_{AllStatesCoverage} = \frac{km}{tm} = \frac{(0+0)}{2} = 0$$

Ovviamente i due 0 rappresentano l'incapacità della test suite di uccidere il mutante 1 e il mutante 2. Passando alla test suite relativa al criterio All-Transitions Coverage, essa è in grado di uccidere solo il mutante 1, di conseguenza il calcolo dell'efficacia è effettuato come segue:

$$TER_{AllTransitionsCoverage} = \frac{km}{tm} = \frac{(1+0)}{2} = 0.5$$

In ultimo si calcola l'efficacia relativa alla test suite All-One-Loop-Paths Coverage, che è in grado di uccidere entrambi i mutanti. Per tale ragione il calcolo dell'efficacia dà il seguente risultato:

$$TER_{AllOneLoopPathsCoverage} = \frac{km}{tm} = \frac{(1+1)}{2} = 1$$

Tale confronto fornisce nuove informazioni circa la test suite da ritenere migliore, infatti nonostante la test suite di All-Transitions Coverage abbia una copertura maggiore rispetto a quella del criterio All-One-Loop-Paths Coverage, è in grado solo di uccidere uno dei due mutanti. Per tale ragione la domanda da porsi è se è possibile aumentare la copertura della test suite All-One-Loop-Paths Coverage.

La risposta a questo quesito è affermativa e anche piuttosto semplice, infatti non bisogna far altro che aggiungere i test case di figura 7.98 a uno dei tre script della test suite All-One-Loop-Paths Coverage. In figura 7.115 si mostra l'analisi relativa alla nuova copertura raggiunta.

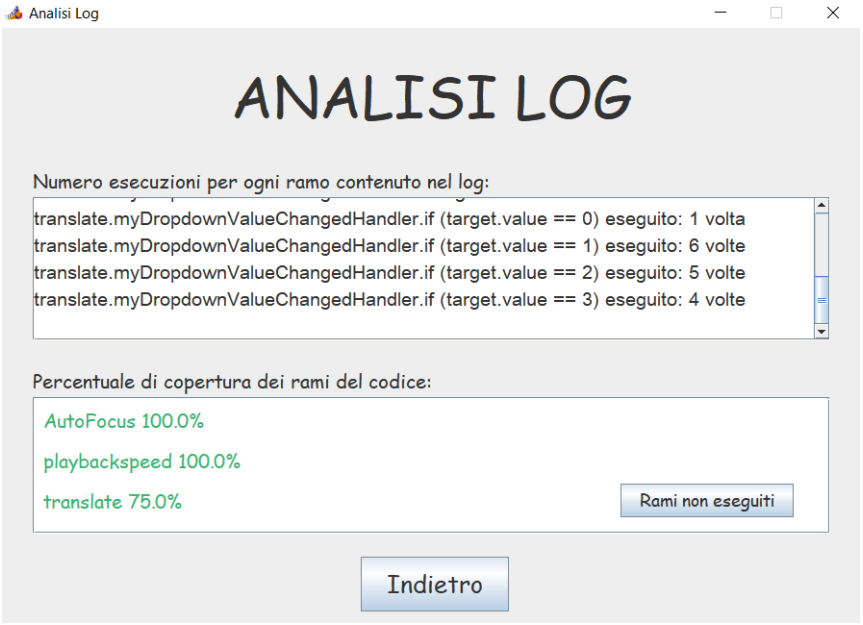


Figura 7.115: Nuova analisi All-One-Loop-Path Coverage

In definitiva, la test suite All-One-Loop-Paths Coverage presenta una copertura pari a quella del criterio All-Transitions Coverage, ma con un'efficacia maggiore. Per tali motivazioni si conclude essere la migliore test suite per il testing dell'applicazione in esame.

	AutoFocus	**translate**	**playbackspeed**
Test suite All-One-Loop-Paths Coverage	100% $(\frac{4}{4})$	75% $(\frac{6}{8})$	100% $(\frac{1}{1})$

Tabella 7.46: Nuova percentuale copertura

Capitolo 8

Conclusioni e Sviluppi futuri

Con questo lavoro di tesi si è cercato di adattare i tools utilizzati per il testing di applicazioni mobile ad applicazioni AR mobile. Quanto mostrato nei capitoli precedenti, relativamente all'effettuazione dei test, è riassunto nel repository GitHub, raggiungibile da [24]. Come visto, i tools che più è stato possibile adattare allo scopo sono stati quelli messi a disposizione da AirTestProject, nella fattispecie Poco e AirTestIDE. I risultati raggiunti sono stati discreti automatizzando, anche solo parte, il testing delle GUI di questo tipo di applicazioni, evitando così a tester umani di compiere azioni ripetitive e che potrebbero risultare noiose. La motivazione, che ha spinto a cercare una soluzione di questo genere, è stata quella di valutare l'effettiva fattibilità del testing automation con le tecnologie oggi presenti ponendosi l'obiettivo

di ottimizzare i tempi di testing mantenendo sempre la stessa precisione. Tuttavia siamo ancora molto lontani dall'avere uno strumento in grado di verificare la correttezza in toto delle applicazioni AR. Infatti restano ancora problematiche irrisolte quali: inserimento di input sonori (ad esempio comandi vocali), verifica delle animazioni degli oggetti presenti sulla scena, verifica di segnali audio di output o ancora l'accesso a tutte le componenti dei GameObject. Queste limitazioni non ci permettono, ad oggi, di poter definire i tools di AirTestProject come lo standard con cui dover testare le applicazioni mobile di realtà aumentata. Tuttavia essendo che è un progetto ancora abbastanza nuovo e in fase di sviluppo, si confida che con l'aumentare della diffusione di questa gamma di applicazioni, vengano aggiunte al progetto nuove funzionalità che permettano di superare questa tipologia di problematiche. L'orizzonte temporale in cui si confida per l'aggiunta delle nuove funzionalità è l'immediato futuro che, a mio avviso, coinciderà con la pervasività della realtà aumentata nella vita di tutti i giorni, proiettandoci in quella che sarà la nuova quotidianità dei prossimi decenni. Per quanto riguarda gli sviluppi futuri di tale lavoro di tesi sono principalmente due:

- Estensione del programma "Analisi Log";
- Generazione automatica del codice a partire dal modello creato.

Per quanto riguarda il primo punto, aggiungere la funzionalità di inserimento automatico delle sonde, all'interno degli script del progetto

da testare, ridurrebbe notevolmente il tempo necessario al tester per svolgere tale mansione. Oltre a ciò, avendo prefissato un template, si potrà essere sicuri che tali sonde siano state inserite correttamente senza errori di distrazione. Mentre per il secondo punto, trovare un tool che generi il codice di test a partire dal modello creato, assicurerebbe, anche in questo caso, una notevole ottimizzazione dei tempi grazie alla generazione di codice già strutturato. Un possibile tool in grado di fare ciò potrebbe essere GraphWalker [26], che dovrebbe essere integrato ai succitati tools.

Bibliografia

[1] https://altom.com/testing-tools/altunitytester. acceduto il 12-01-2021.

[2] https://github.com/AirtestProject/Poco. acceduto il 31-01-2021.

[3] https://www.oxfordlearnersdictionaries.com/definition/english/augmented-reality?q=augmented+reality. acceduto il 16-12-2020.

[4] https://www.sei.cmu.edu/architecture/start/glossary/community.cfm. acceduto il 19-12-2020.

[5] https://developer.apple.com/arkit. acceduto il 23-12-2020.

[6] https://developers.google.com/ar. acceduto il 27-12-2020.

[7] https://www.wikitude.com/external/doc/expertedition. acceduto il 27-12-2020.

[8] http://www.hitl.washington.edu/artoolkit/documentation. acceduto il 27-12-2020.

[9] https://library.vuforia.com. acceduto il 27-12-2020.

[10] https://developer.vuforia.com. acceduto il 30-12-2020.

[11] https://assetstore.unity.com. acceduto il 30-12-2020.

[12] https://unity3d.com/get-unity/download. acceduto il 30-12-2020.

[13] https://altom.gitlab.io/altunity/altunitytester/pages/overview.html#key-features. acceduto il 12-01-2021.

[14] https://airtest.doc.io.netease.com/en. acceduto il 12-01-2021.

[15] https://github.com/AirtestProject/Poco-SDK. acceduto il 12-01-2021.

[16] https://airtest.doc.io.netease.com/en/IDEdocs/airtest_framework/0_airtest_api_info. acceduto il 12-01-2021.

[17] https://airtest.netease.com. acceduto il 12-01-2021.

[18] https://altom.gitlab.io/altunity/altunitytester/pages/get-started.html. acceduto il 12-01-2021.

[19] https://poco-chinese.readthedocs.io/zh_CN/latest/source/doc/integration.html#unity3d%3E. acceduto il 12-01-2021.

[20] https://github.com/search?o=desc&p=1&q=Unity+Vuforia+augmented-reality+app&s=stars&type=Repositories. acceduto il 26-01-2021.

[21] https://github.com/search?q=Unity+Vuforia+ar+app&type=Repositories. acceduto il 27-01-2021.

[22] https://www.pikkart.com/contenuto/contenuti--ecm/pikkart-ar-musei.ashx. acceduto il 14-02-2021.

[23] https://github.com/abdullahibneat/SafariAnimalsAR. acceduto il 29-01-2021.

[24] https://github.com/danilobevilacqua/TesiMagistrale. acceduto il 11-04-2021.

[25] https://github.com/abdullahibneat/PointAR. acceduto il 17-03-2021.

[26] https://github.com/GraphWalker/graphwalker-project. acceduto il 29-04-2021.

[27] Siti Sendari, Adim Firmansah, Aripriharta. Performance analysis of augmented reality based on vuforia using 3d marker detection.

2020 4th International Conference on Vocational Education and Training (ICOVET), Settembre 2020.

[28] Ronald T. Azuma. A survey of augmented reality. *Teleoperators and Virtual Environments*, Agosto 1997.

[29] Partha Sarathi Paul, Surajit Goon, Abhishek Bhattacharya. History and comparative study of modern game engines. *International Journal of Advanced Computer and Mathematical Sciences*, Gennaio 2012.

[30] Robert V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley Professional, Ottobre 1999.

[31] Asa MacWilliams, Thomas Reicher, Bernd Bruegge. Study on software architectures for augmented reality systems. *ARVIKA*, Ottobre 2002.

[32] Asa MacWilliams, Thomas Reicher, Gudrun Klinker, Bernd Bruegge. Design patterns for augmented reality systems. *MIXER*, Gennaio 2004.

[33] Carmen Juan, Francesca Beatrice, Juan Cano. An augmented reality system for learning the interior of the human body. *Eighth IEEE International Conference on Advanced Learning Technologies*, Luglio 2008.

[34] Sergio Martin, Gerardo Parra, Joaquín Cubillo, Blanca Quintana, Rosario Gil, Clara Perez, Manuel Castro. Design of an augmented reality system for immersive learning of digital electronic. *2020 XIV Technologies Applied to Electronics Teaching Conference (TAEE)*, Luglio 2020.

[35] Jae-Young Lee, Seok-Han Lee, Hyung-Min Park, Sang-Keun Lee, Jong-Soo Choi. Design and implementation o f a wearable ar annotation system using gaze interaction. *Digest of Technical Papers International Conference on Consumer Electronics (ICCE)*, Gennaio 2010.

[36] Richard N. Taylor, Nenad Medvidovic, Eric Dashofy. *Software Architecture: Foundations, Theory, and Practice.* John Wiley & Sons, Gennaio 2009.

[37] Steven J. Henderson, Steven K. Feiner. Augmented reality in the psychomotor phase of a procedural task. *10th IEEE International Symposium on Mixed and Augmented*, Ottobre 2011.

[38] Mark Grechanik, Qing Xie, Chen Fu. Creating gui testing tools using accessibility technologies. *International Conference on Software Testing, Verification, and Validation Workshops*, Maggio 2009.

[39] Mark Grechanik, Qing Xie, Chen Fu. Maintaining and evolving gui-directed test scripts. *IEEE 31st International Conference on Software Engineering*, Maggio 2009.

[40] Dhiraj Amin, Sharvari Govilkar. Comparative study of augmented reality sdk's. *International Journal on Computational Science & Applications*, Febbraio 2015.

[41] Jens Grubert, Raphael Grasset. *Augmented Reality for Android Application Development*. Packt Publishing, Novembre 2013.

[42] Elaachak Lotfi, Mohammed Bouhorma, Anasse Hanafi. A comparative study of augmented reality sdks to develop an educational application in chemical field. *The 2nd International Conference on Networking, Information Systems & Security*, Marzo 2019.

[43] The Institute of Electrical and Electronics Engineers, 3 Park Avenue, New York, NY 10016-5997, USA. *IEEE Std 1471*, Settembre 2000.

[44] The Institute of Electrical and Electronics Engineers, 3 Park Avenue, New York, NY 10016-5997, USA. *ISO/IEC/IEEE 29119-1*, Settembre 2013.

[45] Julie Carmigniani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, Misa Ivkovic. Augmented reality technologies,

systems and applications. *Multimedia Tools and Applications*, Dicembre 2010.

[46] Robert V. Binder, Bruno Legeard, Anne Kramer. Model-based testing: Where does it stand? *Communications of the ACM*, Gennaio 2015.

[47] Jesus David Gonzalez, J H Escobar, H Sánchez, Jhon De la Hoz. 2d and 3d virtual interactive laboratories of physics on unity platform. *Journal of Physics Conference Series 935(1):012069)*, Dicembre 2017.

[48] Clemens Arth, Lukas Gruber, Raphael Grasset, Tobias Langlotz. *The History of Mobile Augmented Reality.* Institut für Computer Graphik und Wissensvisualisierung, Technische Universität Graz, Maggio 2015.

[49] Mauricio Hincapié Montoya. Evaluating the effect on user perception and performance of static and dynamic contents deployed in augmented reality based learning application. *Eurasia Journal of Mathematics, Science and Technology Education*, Giugno 2016.

[50] Oliver Bimber, Ramesh Raskar. *Spatial augmented reality: merging real and virtual worlds.* A K Peters LTD, Agosto 2005.

[51] Yasir Dawood, Nor Laily Hashim, Mawarny Md Rejab, Rohaida BT Romli. Coverage criteria for test case generation using

uml state chart diagram. *The 2nd international conference on applied science and technology 2017 (icast'17)*, Ottobre 2017.

[52] Daniel Wagner, Dieter Schmalstieg. Handheld augmented reality displays. *IEEE Virtual Reality Conference*, Marzo 2006.

[53] Assaf Feldman, Emmanuel Munguia Tapia, Sajid Sadi, Pattie Maes, Chris Schmandt. Reachmedia: On-the-move interaction with everyday objects. *Ninth IEEE International Symposium on Wearable Computers*, Ottobre 2005.

[54] Sarah M. Lehman, Haibin Ling, Chiu C. Tan. A user-focused framework for testing augmented reality applications in the wild. *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, Maggio 2020.

[55] Enzo Troisi. *Analisi di Issue, Pull Request e Test in progetti open-source di Realtà Aumentata.* Maggio 2021.

[56] Steven J. Vaughan-Nichols. Augmented reality: No longer a novelty? *Computer*, Dicembre 2009.

[57] Dewayne E. Perry, Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, Ottobre 1992.