UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

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

Elaborato finale in **Ingegneria del Software II**

Strumenti di Capture and Replay in ambito Android

Anno Accademico 2018/2019

Candidato:

Fabio Maresca

matr. N46 002284

Indice

Elenco delle figure

Introduzione

La derivazione manuale dei casi di test può risultare lunga ed onerosa se il sistema studiato è complesso. Il seguente elaborato si propone di presentare delle alternative che riescano ad ovviare a questi problemi; in particolare, tra i vari approcci di generazione automatica di test cases, viene analizzata una tecnica di tipo User Session-based; essa prevede la generazione automatica del codice in seguito all'interazione con il sistema da parte di utenti. Il processo avviene in un ambiente controllato, detto Sistema di Capture; tra i vari si è scelto di utilizzare Espresso Test Recorder, offerto dall'ambiente di sviluppo Android Studio.

Quest'ultimo consente la creazione, lo sviluppo, ed il testing di applicazioni Android; a tal proposito nel primo capitolo viene presentata una panoramica generale del sistema operativo in questione, introducendo brevemente la realtà tecnologica in cui è nato ed i punti di forza che ne hanno permesso lo sviluppo. Nel secondo capitolo viene descritto il funzionamento del tool, approfondendo le fasi di record e replay. Successivamente viene messo in pratica quanto detto, testando alcune applicazioni Android open source. Nel quarto capitolo vengono studiate alcune impostazioni che influiscono sulla generazione automatica del codice. Infine, l'ultimo capitolo evidenzia i punti di forza del tool e le problematiche ad esso associate.

Capitolo 1

Introduzione ad Android

1.1 Una realtà in continua evoluzione

Già da molti anni Internet ha assunto un ruolo sempre più importante nella vita di tutti i giorni, anzi può essere inteso come una rivoluzione non solo tecnologica ma anche culturale: grazie ad esso chiunque è in grado di pubblicare informazioni accessibili da qualsiasi parte del mondo, consentendo una maggiore diffusione di queste e di conseguenza una migliore distribuzione e condivisione della conoscenza. Mentre queste informazioni erano raggiungibili esclusivamente attraverso un computer, ora lo sono anche da dispositivi mobili. Questi, in continua evoluzione, non si limitano più alla funzione base per cui sono stati creati; telefonare, ma riescono a svolgere servizi di vario genere che prima erano pensati esclusivamente per altri dispositvi appositi, seppur con limitazioni e prestazioni minori. E' possibile fotografare, registrare audio, agire direttamente sui risultati di queste azioni modificandone i particolari, condividerli a persone specifiche o su siti accessibili a tutti.

In questa realtà in continua evoluzione nasce la necessità di rendere sempre più performanti sia le applicazioni che i sistemi operativi utilizzati. Per questo motivo sono nate diverse piattaforme, ciascuna con le proprie caratteristiche, il proprio ambiente di sviluppo, ed i propri tool; tra questi ad esempio, iOS per iphone ed ipad utilizza un proprio sistema

operativo Mac OS X e un linguaggio Objective-C, mentre Nokia utilizza Symbian e C++. Google cerca di offrire un'alternativa, lanciando una piattaforma open source, Android. [1]

1.2 Android

Android è un insieme di componenti software, comprende:

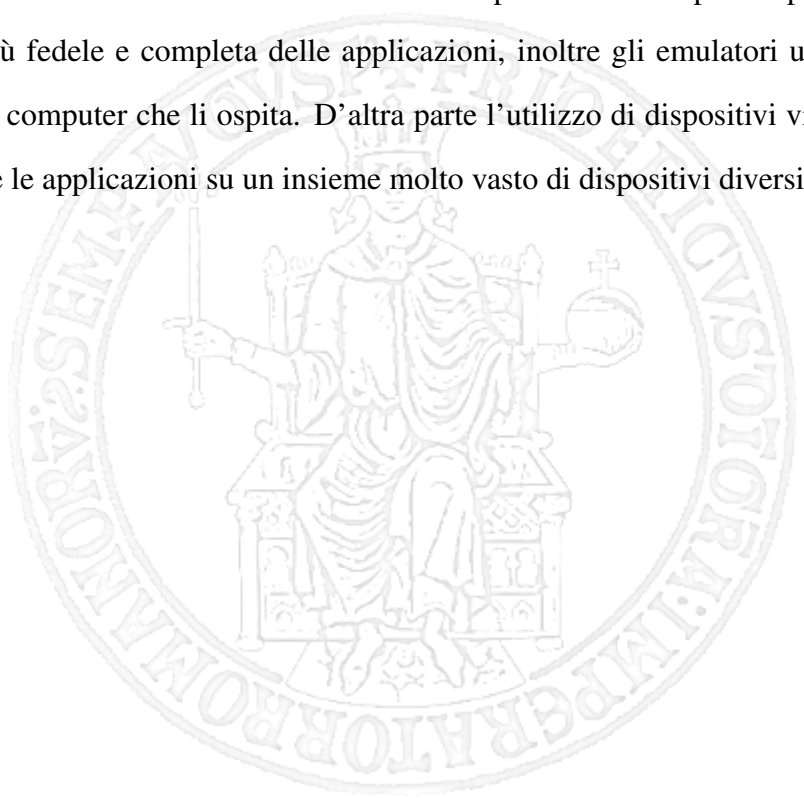
- un sistema operativo, in particolare una distribuzione ridotta di Linux (Kernel 2.6).
- un insieme di librerie ed API utilizzate sia per la realizzazione dell'ambiente che delle applicazioni su cui girano; permettono molte delle features più importanti, come telefonia, messagistica, connettività, touch-screen, storage, multi-tasking, screen-capture.
- un middleware che permette a programmi differenti di comunicare nonostante abbiano protocolli diversi, rivestendo il ruolo di intermediario.

E' utilizzato per una vastissima gamma di dispositivi: cellulari, tablet, oggetti indossabili come smarth-watch, televisori, e supporti ad essi come gli smartBox. E' proprio questo uno dei motivi per cui Android ha riscosso tanto successo: "è stato adottato da dispositivi molto diversi tra loro non solo per tipologia ma anche per fasce di prezzo, da poche decine di euro fino a cifre piuttosto significative; ciò ne ha permesso una diffusione in diverse categorie sociali. Tuttavia questo fenomeno ha costretto di riflesso gli sviluppatori ad adattare il sistema alle caratteristiche del dispositivo ospite." [5]

Android ha riscosso molto successo in pochi anni soprattutto perchè è open source, infatti utilizza un sistema operativo e delle librerie accessibili a tutti; il suo codice è consultabile da chiunque possa contribuire a migliorarlo o voglia semplicemente studiarlo. Rimane open source pur usando Java come linguaggio di programmazione, poichè non necessita della JVM (propria di Java), non dovendo eseguire bytcode Java. Inizialmente offre una propria Virtual Machine (chiamata Dalvik) per ottimizzare l'utilizzo delle applicazioni, in particolare questa esegue il codice contenuto in file di tipo *.dex* ottenuti a

loro volta a partire da file *.class* di bytecode Java. La DVM dapprima è stata affiancata da un'ulteriore virtual machine, ART VM, e successivamente completamente sostituita. Quest'ultima offre una riduzione dei tempi di esecuzione e di conseguenza anche di un corrispondente risparmio energetico grazie alla tecnologia Aehead of Time (AOT) in cui il codice compilato dell'applicazione è direttamente eseguibile, a differenza della tecnologia adottata da Dalvik di tipo Just in Time (JIT). Quindi la scelta di Java come linguaggio di programmazione non va contro la politica open source, anzi presenta diversi vantaggi, tra i quali il non dover realizzare un nuovo compilatore, un debugger, delle librerie idonee, e la documentazione necessaria associata; un ulteriore vantaggio è che gli sviluppatori non sono obbligati ad imparare un nuovo linguaggio di programmazione.

Android fornisce un SDK (Standard Developement Kit) per facilitare lo sviluppo delle applicazioni, questo contiene tutti gli strumenti fondamentali per lo sviluppo di applicazioni java tra cui vari strumenti di supporto e documentazione, un emulatore che permette di eseguire le applicazioni su device virtuali; è infatti possibile eseguire le applicazioni sia su dispositivi reali che virtuali. L'utilizzo dei primi conviene perchè permette un esecuzione più fedele e completa delle applicazioni, inoltre gli emulatori utilizzano molte risorse del computer che li ospita. D'altra parte l'utilizzo di dispositivi virtuali consente di eseguire le applicazioni su un insieme molto vasto di dispositivi diversi.



Capitolo 2

Strumenti di capture and replay

2.1 Testing e Casi di test

Il testing è il processo che esegue il software con lo scopo di scovare malfunzionamenti, il programma viene eseguito con parametri particolari, detti Test Case. La loro progettazione mira a scoprire una classe di errori con il minimo sforzo.

Un primo approccio per la derivazione dei casi di test è dato dal Category Partition Testing, un tipo di test combinatoriale. Esso prevede che le specifiche del sistema vengano strutturate in un insieme di funzionalità testabili separatamente, si individuano i parametri da cui queste dipendono e si stabiliscono dei vincoli per identificare le combinazioni non valide, diminuendo così il numero dei casi di test. Al crescere della complessità e della grandezza del sistema, aumenta anche il numero dei test case e la loro derivazione manuale può diventare lunga, ripetitiva e onerosa.

2.2 Tecniche di generazione automatica

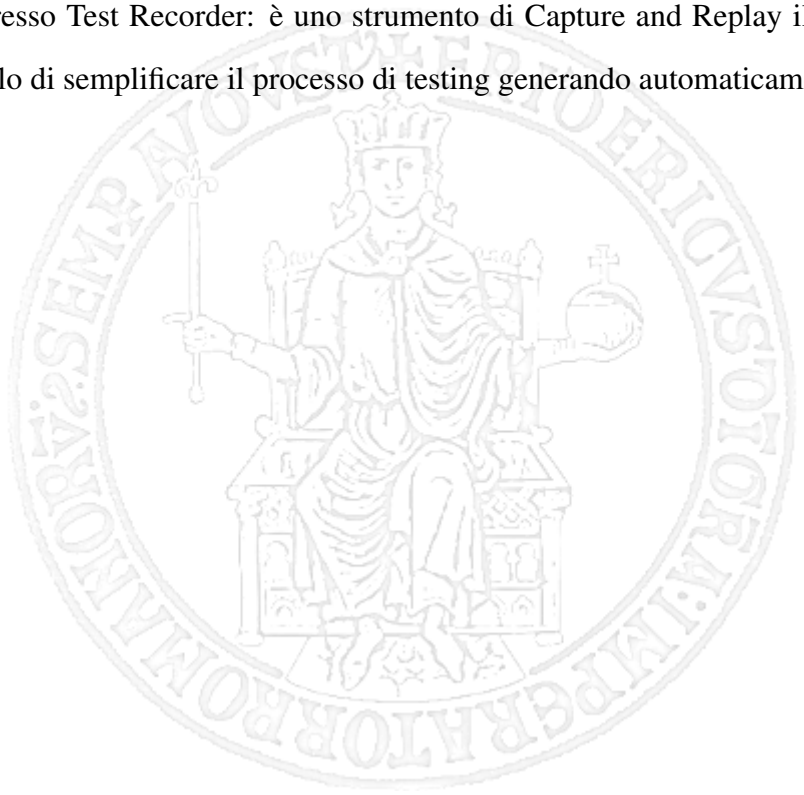
Per ovviare a questi problemi e ridurre tempi e costi legati alla generazione dei casi di test, è possibile derivarli automaticamente tramite l'utilizzo di tecniche apposite; ad esempio analizzando la documentazione di analisi o di progetto, analizzando il codice sorgente, oppure con l'ausilio di un modello del sistema. E' il caso del Model Based Testing: il modello è una descrizione più semplice del sistema che aiuta a comprenderne il funzionamento. In questo modo è possibile predire il suo comportamento a runtime e le risposte a determinate azioni. Un modello inesatto o approssimativo del sistema può comportare delle previsioni errate, d'altra parte un modello esaustivo spesso è oneroso.

Altre tecniche di generazione automatica dei casi di test prevedono l'interazione con l'applicazione in esecuzione, in maniera sistematica o casuale. In questi ultimi casi si parla di User Session based Testing, una tecnica black box basata sull'analisi di input immessi e output ottenuti durante l'esecuzione dell'applicazione. Ad interagire con l'applicazione sono gli utenti, il cui approccio può variare in base all'esperienza e al loro ruolo: tester interni all'azienda hanno il compito di condurre un'analisi sistematica dell'applicazione per coprire più casi d'uso possibili, mentre utenti esterni potrebbero comportarsi in maniera naturale e utilizzare solo una parte delle funzionalità dell'applicazione, non preoccupandosi delle altre più specifiche che non rientrano nei loro interessi. L'interazione può avvenire in un ambiente controllato, detto Sistema di Capture, che registra le azioni effettuate dall'utente in un log. Successivamente queste vengono tradotte automaticamente in codice eseguibile, ottenendo così un caso di test che può essere rieseguito o essere usato come oracolo per futuri replay. Nella fase di replay il codice ottenuto viene eseguito, e le azioni registrate nella fase precedente vengono replicate in maniera fedele.

2.3 Espresso Test Recorder:

Android Studio offre diversi strumenti a supporto di creazione, studio e testing di applicazioni Android. Precedentemente si utilizzavano estensioni di Eclipse, fin quando Google ha adattato ad Android l'IDE *IntelliJ Idea*, costruendo il nuovo ambiente di sviluppo integrato (IDE) Android Studio. Esso è disponibile per i sistemi operativi più comuni, tra i quali Windows, Linux e Mac OS X. I tools utili a test di tipo User Session Based sono:

- AVD Manager: consente di creare emulatori diversi sia per caratteristiche hardware che software. E' infatti possibile scegliere uno tra più modelli di cellulari o tablet e la versione del sistema operativo. Inizialmente sono proposti solo alcuni dei brand che utilizzano Android, ma è possibile importarne altri. Analogamente per la versione del sistema operativo, è possibile scaricare una API cliccando sulla scritta "Download" presente accanto alle versioni non ancora ottenute. La creazione di una macchina virtuale tramite AVD Manager avviene in tre passaggi, illustrati nella figura 2.3.1.
- Espresso Test Recorder: è uno strumento di Capture and Replay il cui compito è quello di semplificare il processo di testing generando automaticamente Test Case.



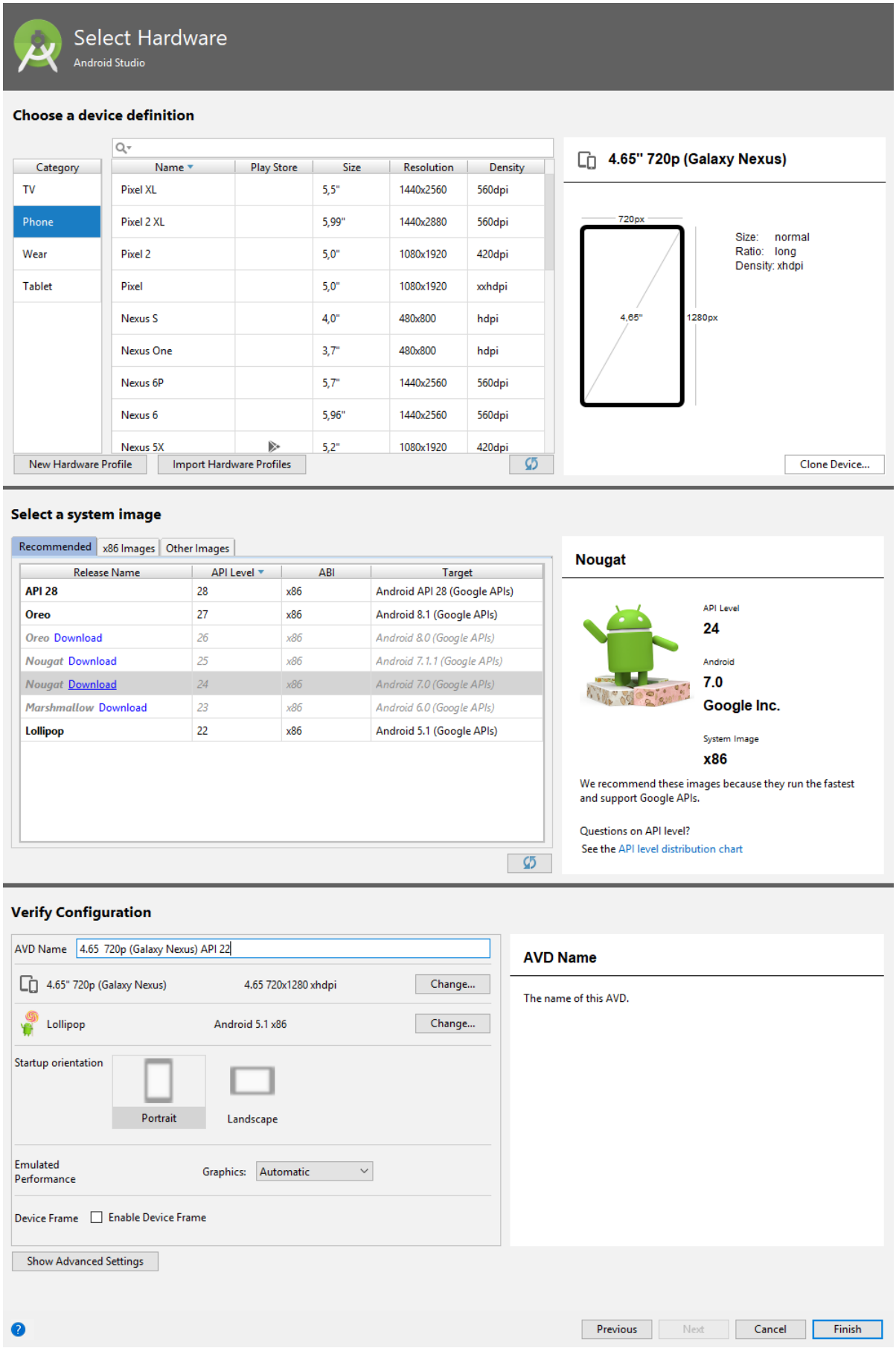


Figura 2.3.1: passaggi per la creazione di un emulatore con AVD Manager

2.3.1 Descrizione del tool

I concetti fondamentali su cui si basa sono le UI interactions e le asserzioni. Le interazioni rappresentano tutte le azioni che una persona potrebbe eseguire utilizzando normalmente l'applicazione; le asserzioni, invece, servono a verificare la presenza di elementi sull'interfaccia ed il loro stato. Espresso permette quindi di interagire con l'applicazione in ambiente controllato e di creare UI tests senza scrivere codice, in particolare: registra tutte le interazioni con gli elementi dell'interfaccia, successivamente genera automaticamente un caso di test che corrisponde alle azioni effettuate. Questo può essere riprodotto per verificare l'esatto funzionamento dell'applicazione.

Per poter utilizzare Espresso in Android Studio occorre cliccare su *Run* e selezionare la voce *Record Espresso Test* come mostrato in figura. Successivamente bisogna scegliere il tipo di device da utilizzare per il test, tra cui le macchine virtuali precedentemente create o eventuali dispositivi reali collegati tramite USB, è possibile creare un nuovo emulatore anche da questa schermata.

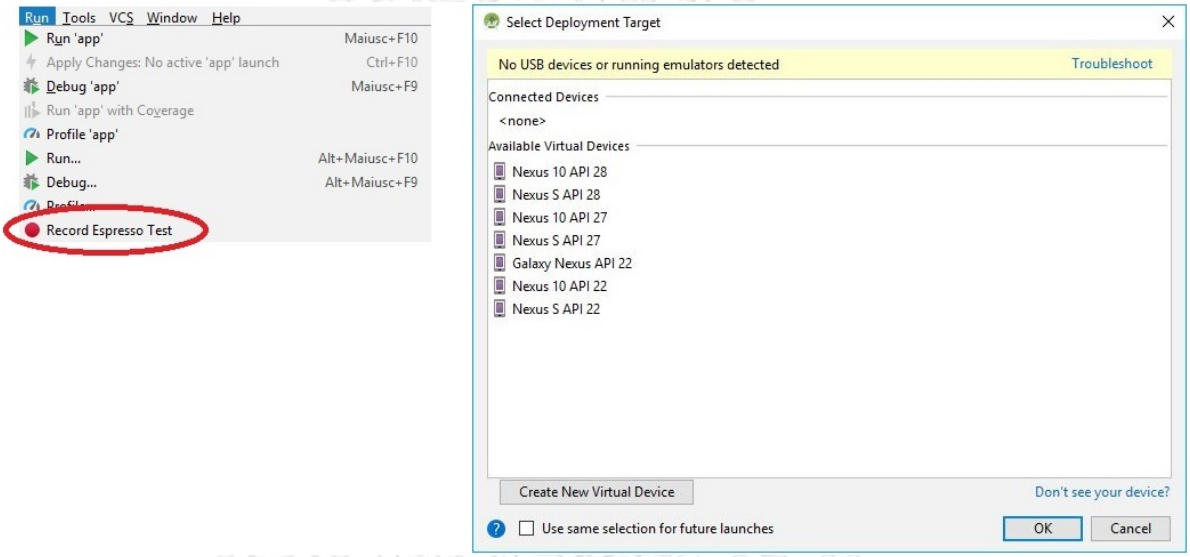


Figura 2.3.2: a) Android Studio -> run -> Record Espresso Test.
b) Scelta di un device connesso tramite USB, o di un emulatore.

2.3.2 Fase di Capture

Una volta selezionato il dispositivo, e terminate le azioni di preparazione necessarie all'avvio dell'applicazione (build del progetto ed installazione della stessa sul dispositivo), comparirà la finestra *Record Your Test*. Il messaggio "No events recorded yet" viene sostituito dall'elenco delle azioni registrate nel momento in cui esse sono eseguite. *Add assertions* permette di verificare l'esistenza di determinati elementi dell'interfaccia o il contenuto di essi attraverso delle asserzioni. Espresso crea una vista dell'interfaccia dell'applicazione ed aiuta il tester a riconoscere i vari elementi evidenziandoli con dei riquadri rossi; è possibile selezionarli cliccandoli sulla vista oppure scegliendoli dalla lista del menù *Edit Assertion*. Il tipo di asserzione varia in base all'elemento selezionato (ad esempio un pulsante può esistere o meno, e quindi l'asserzione sarà *Assert Button with ID X exists/does not exist*; per le sezioni caratterizzate da testo è inoltre possibile asserire *Text is* e controllare cosa vi sia scritto, ad esempio per una calcolatrice è possibile asserire *Assert Button with ID result text is 10* in seguito ad un operazione 6 + 4). Durante la creazione di un'asserzione, è possibile continuare ad interagire con l'applicazione nonostante la vista sia ancora aperta: le azioni vengono scritte normalmente nel log, ma una volta ultimata l'asserzione, questa verrà posizionata nel punto in cui è stata iniziata, ovvero prima delle interazioni.

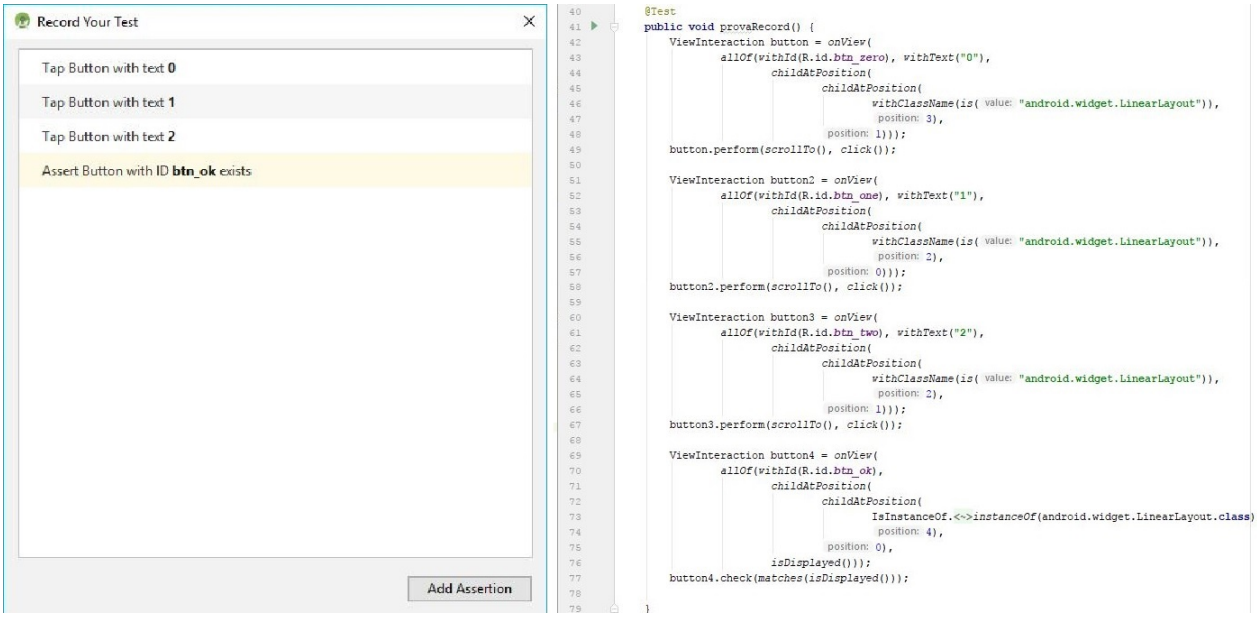


Figura 2.3.3: a) Registrazione nel log delle azioni effettuate.
b) Codice generato relativo alla figura 2.3.3a.

Nel codice generato, gli oggetti della GUI coinvolti nel record sono identificati da un ID, spesso anche dal testo con cui vengono mostrati sull'interfaccia, o anche da quali classi derivano (ChildOfPosition). Per riferirsi ad un oggetto di una determinata vista dell'interfaccia si utilizza il metodo *onView( )*, questo ritorna un oggetto ViewInteraction che permette di interagire con la vista. Per indicare da che tipo di attributo esso sia identificato si utilizza il metodo *with*( )*, dove * sta per il tipo di attributo; ad esempio per Id e testo si utilizzano rispettivamente *withId( )* e *withText( )*. Inoltre è possibile specificare se un oggetto sia caratterizzato da più di un attributo, e considerarli tutti con il metodo *allOf( )*. *perform( )* simula le interazioni utente sulla UI, gli argomenti necessari della funzione sono uno o più oggetti ViewAction. La classe ViewAction presenta una lista di metodi relativi alle azioni più comuni, come il click del mouse con *click( )*, scrivere testo in aree apposite con *typeText( )*, cancellare il testo con *clearText( )*, scorrere la vista con *scrollTo( )*. Infine i metodi della classe ViewAssertions sono usati per controllare che l'interfaccia rispetti lo stato desiderato in seguito alle azioni effettuate.

2.3.3 Fase di Replay

Per rieseguire le azioni effettuate in fase di record, è necessario selezionare il caso di test creato automaticamente ed avviarlo su un dispositivo reale oppure su una macchina virtuale preesistente; eventualmente è possibile crearne una al momento dell'avvio. Android studio effettua una build del progetto; nella finestra Run è possibile monitorare l'andamento del test, l'esito ed il tempo impiegato, fino ad arrivare al messaggio "Tests ran to completion" nel momento in cui il test è finito correttamente. Durante il replay le azioni vengono eseguite nello stesso ordine con cui sono state registrate, ma non con le stesse tempistiche.

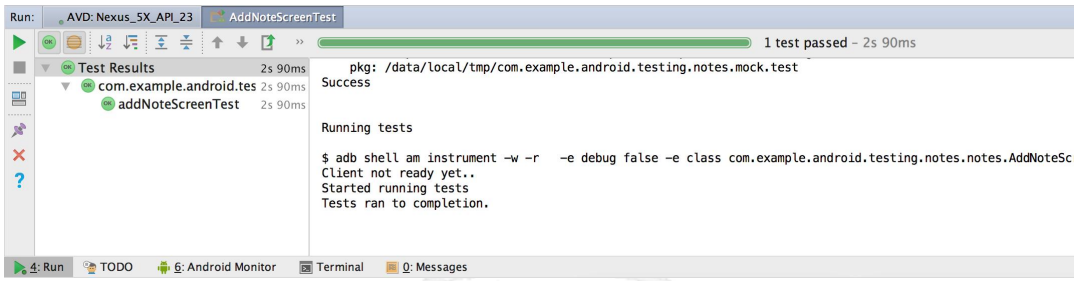


Figura 2.3.4: andamento dell'esecuzione del caso di test.

Capitolo 3

Esempi di applicazioni testate

3.1 Tippy Tipper

"TippyTipper" è un'applicazione open source scaricabile dal market Google Play, presenta una seconda versione, identica a quella di base ("donate version"), acquistabile per supportare lo sviluppatore. TippyTipper è una calcolatrice che ha come obiettivo quello di calcolare la mancia (tip) in funzione del prezzo totale da pagare in un ristorante.

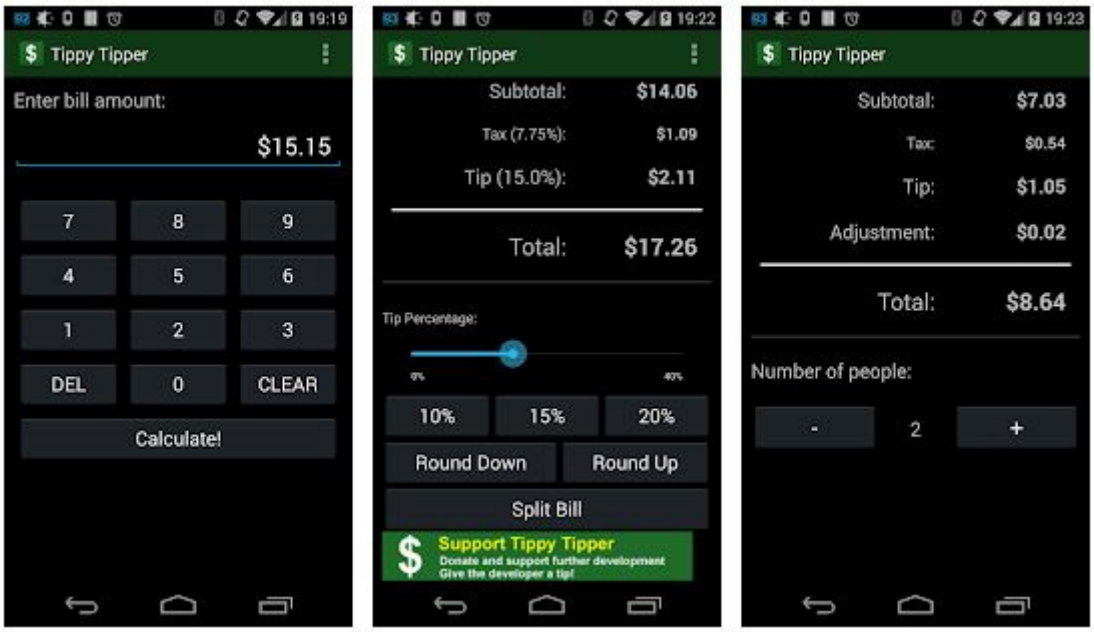


Figura 3.1.1: Anteprima interfaccia dell'applicazione.

L'interfaccia è composta dai tipici widget di una calcolatrice virtuale, quindi da una sezione dello schermo in cui sono riportate le cifre digitate, da una tastiera numerica che comprende anche tasti per l'eliminazione di una o tutte le cifre inserite (rispettivamente DEL e CLEAR) e di un tasto per calcolare la mancia una volta inserito l'importo (CALCULATE). Una volta premuto quest'ultimo, è possibile variare la percentuale interagendo con dei pulsanti o regolando il livello di una barra. E' possibile inoltre arrotondare per difetto o per eccesso, e suddividere le spese tra più persone cliccando su "Split Bill".

3.1.1 Fase di Capture

Nella fase di capture, Espresso Test Recorder recepisce l'interazione sia con pulsanti dell'interfaccia proprie dell'applicazione, che con funzionalità dell'emulatore esterne ad essa, e le registra. Tuttavia non tutte sono interpretate nel modo corretto. Ad esempio, viene utilizzato un emulatore *Nexus S* (API 27, corrispondente alla versione Oreo 8.1), e vengono effettuate le seguenti azioni:

- premere i seguenti tasti in successione: 1, 0, 0
- girare il cellulare interagendo con il pulsante di rotazione dell'emulatore
- premere i seguenti tasti in successione: 1, 0, 0, 9, 9
- premere il tasto CALCULATE
- impostare la percentuale della mancia a 10%, successivamente a 20%, utilizzando gli appositi tasti
- impostare la percentuale della mancia a 30% regolando il livello della barra
- premere il tasto SplitBill e successivamente +
- ritornare all'interfaccia originaria attraverso il pulsande dell'emulatore Back

Queste verranno così registrate nel log:

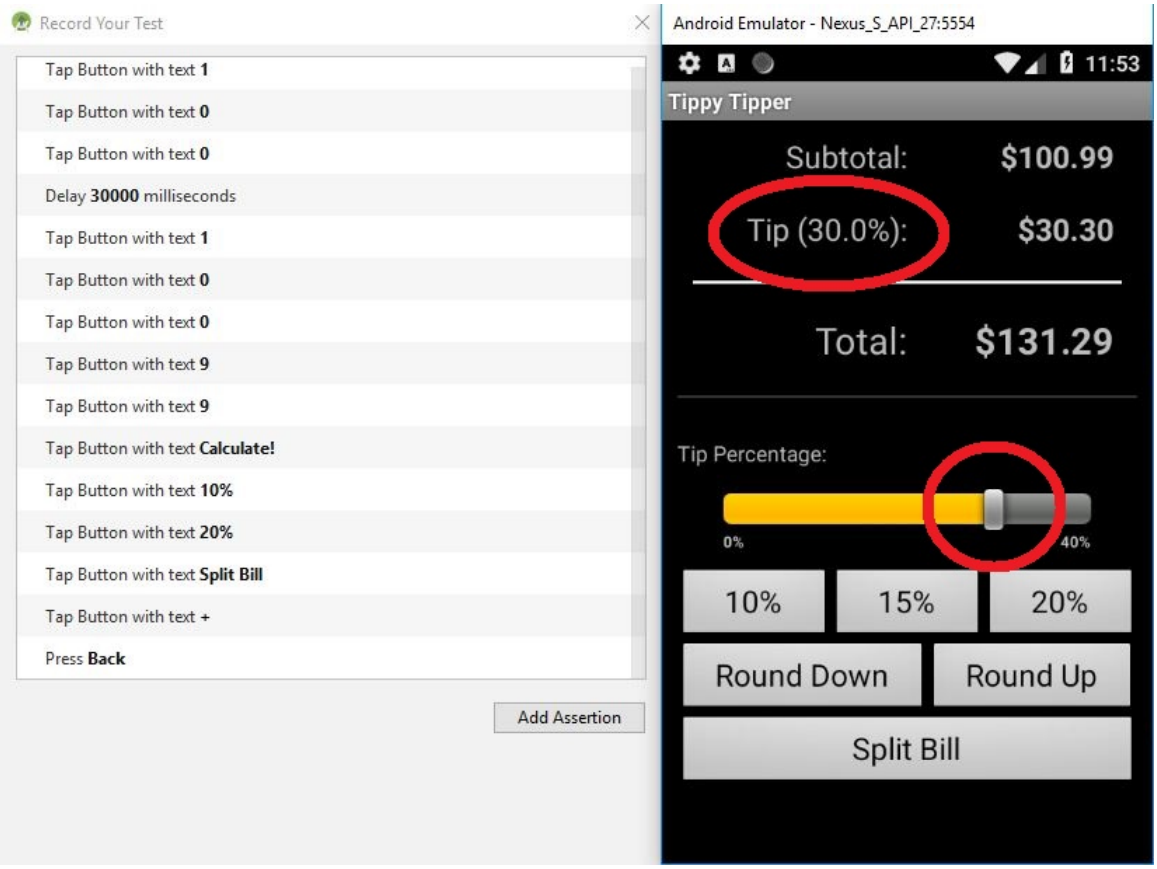


Figura 3.1.2: a) Registrazione azioni. b) Interazione con la barra della percentuale.

Confrontando le azioni effettuate con ciò che compare nel log, è possibile osservare che tutte le azioni relative ai pulsanti dell'interfaccia vengono interpretate correttamente; ciò non accade con lo scorrimento della barra orizzontale per la regolazione della percentuale. Questa azione non viene registrata, di conseguenza non sarà presente nel codice del test case e non verrà replicata nella fase di replay. In figura è evidenziato in rosso come la barra indichi 30%, percentuale confermata anche sopra in maniera numerica, mentre nel log del Recorder è fissata a 20%. Con questa sequenza di azioni è possibile notare anche un'altra anomalia: nel momento in cui viene ruotato l'emulatore, l'applicazione crasha improvvisamente; questa interruzione è inaspettata poichè non avviene se l'applicazione viene eseguita al di fuori di Espresso. Il tool registra "Delay 30.000 milliseconds", e continua a registrare nel momento in cui l'applicazione viene riavviata.

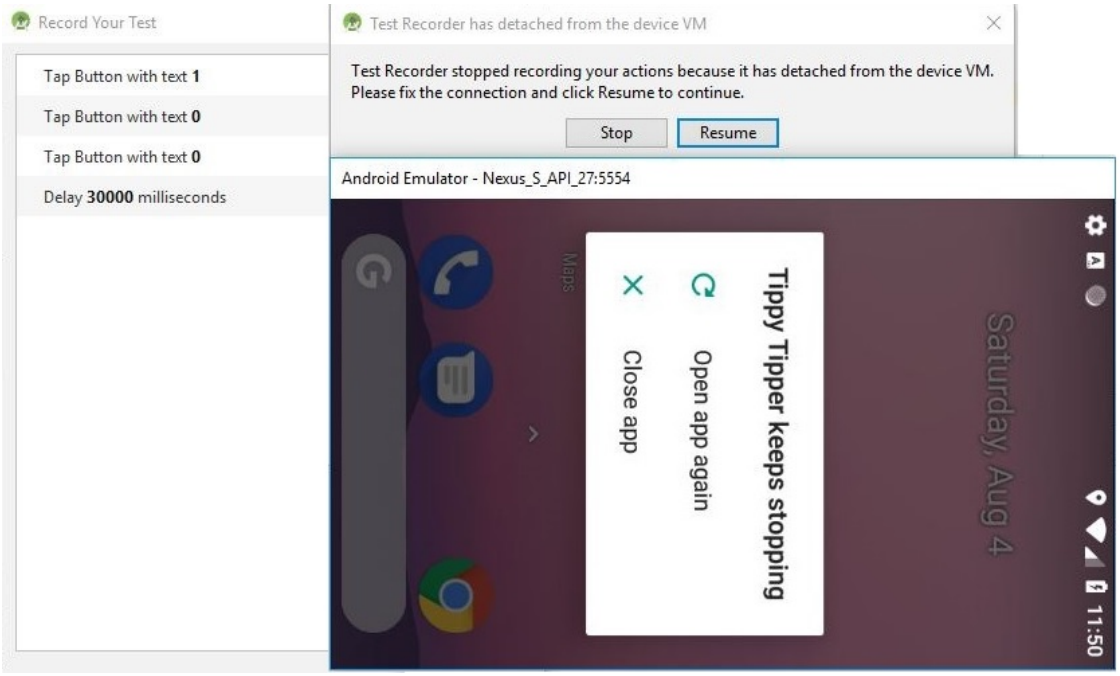


Figura 3.1.3: arresto improvviso dell'applicazione

In figura è mostrato parte del codice generato automaticamente:

```
40  public void tippyProva2() {
41      ViewInteraction button = onView(
42              allOf(withId(R.id.btn_one), withText("1"),
43                      childAtPosition(
44                              childAtPosition(
45                                      withClassName(is( value: "android.widget.LinearLayout")),
46                                       position: 2),
47                              position: 0)));
48      button.perform(scrollTo(), click());
49
50
51      ViewInteraction button2 = onView(
52              allOf(withId(R.id.btn_zero), withText("0"),
53                      childAtPosition(
54                              childAtPosition(
55                                      withClassName(is( value: "android.widget.LinearLayout")),
56                                       position: 3),
57                              position: 1)));
58      button2.perform(scrollTo(), click());
```

```
68
69      // Added a sleep statement to match the app's execution delay.
70      // The recommended way to handle such scenarios is to use Espresso idling resources:
71      // https://google.github.io/android-testing-support-library/docs/espresso/idling-resource/index.html
72      try {
73          Thread.sleep( time: 30000);
74      } catch (InterruptedException e) {
75          e.printStackTrace();
76      }
```

Figura 3.1.4: Individuazione dei pulsanti 1 e 0, ordine in cui devono essere premuti. Gestione del delay attraverso la sleep

3.1.2 Fase di Replay

Durante la fase di replay viene eseguito il test case: così come si era ipotizzato, sono replicate correttamente le azioni che coinvolgono i pulsanti, mentre l'interazione con la barra della percentuale non viene eseguita.

E' possibile notare che, in seguito ai 30 secondi di delay dovuti al crash dell'applicazione, il replay continua in modo errato: nella fase di record l'applicazione segna un importo predefinito di 0.00 dopo essersi riavviata; tuttavia il reset di questo numero è avvenuto a causa di un riavvio forzato dell'applicazione, azione non interpretabile da Espresso, e non a causa della pressione del pulsante CLEAR. Di conseguenza, nel replay vengono aggiunte in successione le cifre 1, 0, 0, 9, 9 a partire da un importo iniziale di 1.00, cifra non resettata, ottenendo un risultato diverso da quello calcolato in fase di record (I risultati sono: 100100.99 partendo da 1.00 nel replay, 100.99 partendo da 0.00 durante l'interazione con l'applicazione). Il replay non può essere effettuato su un qualsiasi dispositivo diverso da quello su cui è stato fatto il record, ad esempio: è stato avviato Espresso Test Record utilizzando la macchina virtuale "Nexus S API 27", mentre sono stati provati i replay su diversi dispositivi. Di seguito una lista degli emulatori coinvolti nell'esperimento e i conseguenti esiti:

- Nexus S API 27, stesso emulatore: il replay avviene correttamente.
- Nexus S API 22, stesso device, ma con sistema operativo meno aggiornato: il replay avviene correttamente.
- Nexus S API 28, stesso device, ma con versione successiva del sistema operativo: in un primo momento il replay non avviene e l'applicazione riporta il seguente messaggio di errore "This app was built for an older version of Android and may not work properly. Try checking for updates, or contact the developer." (figura 1.4.5); il replay viene effettuato correttamente una volta aggiornata l'applicazione.
- Galaxy Nexus API 22, device diverso (cellulare): il replay avviene correttamente.
- Nexus 10, device diverso (tablet): il replay non è attuabile, indipendentemente dalla versione utilizzata, probabilmente a causa delle dimensioni del device dissimili a

quelle su cui è stato effettuato il record, o a causa del fatto che lo schermo del Nexus S è verticale mentre quello del Nexus 10 è orizzontale.

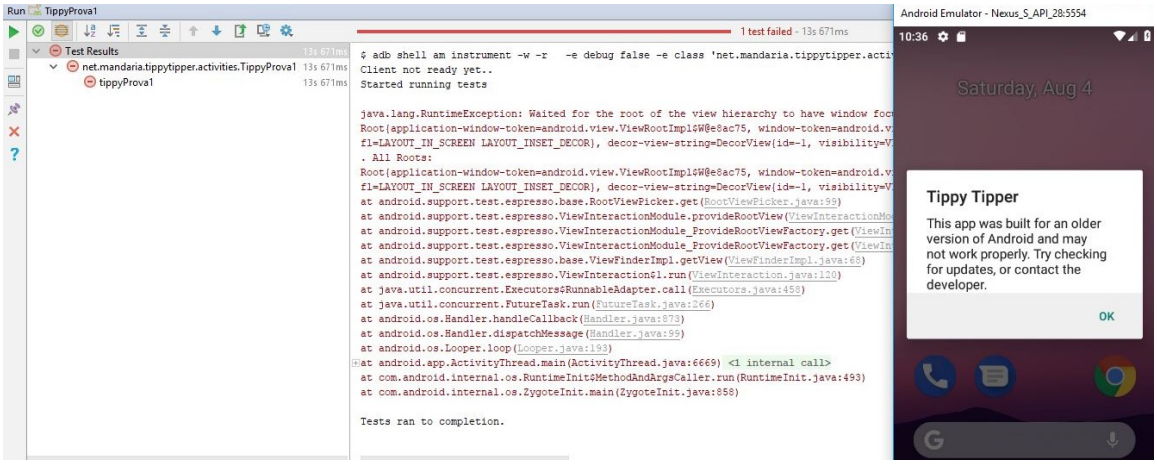


Figura 3.1.5: Replay effettuato su una versione di Android successiva.

3.2 Simple Calendar

Simple Calendar è un'applicazione open source scaricabile da Google Play, il codice è reperibile presso GitHub al seguente link: https://github.com/SimpleMobileTools/Simple-Calendar. Questa agenda virtuale permette di organizzare gli impegni per giorno e orario: è possibile memorizzare un nuovo evento, modificare o cancellarne uno già esistente. Ogni avvenimento è caratterizzato da molti campi opzionali, tra cui titolo, luogo, descrizione, fascia oraria. L'utente può decidere quanti minuti prima essere avvisato, e classificare gli eventi attraverso delle etichette colorate che ne indicano il tipo. Gli avvenimenti registrati sono visualizzabili attraverso più viste: la classica mensile in stile calendario, annuale, settimanale, giornaliera, o anche in una semplice lista.

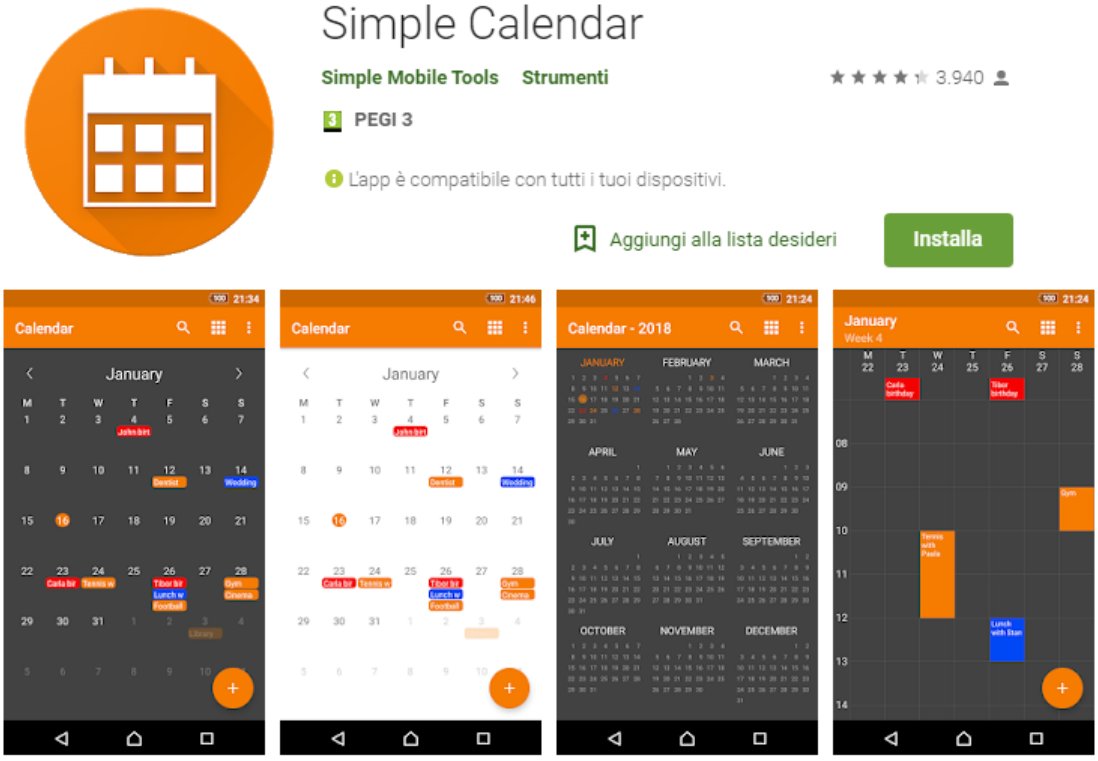


Figura 3.2.1: anteprima dell'applicazione nel play store

L'anteprima di GooglePlay anticipa che è possibile personalizzare i widget; nelle impostazioni è possibile infatti modificare i colori dello sfondo e dei caratteri nella voce *Customize colors*. Inoltre è possibile scegliere se visualizzare l'ora secondo il formato da 12 o 24, ed impostare come il device debba notificare un avvenimento (vibrando o attraverso una tra un'ampia scelta di suonerie). A questo punto l'applicazione chiederà di supportare lo sviluppatore attraverso il messaggio "Please purchase Simple Thank You to unlock this function and support the development, thanks!", ma è possibile effettuare modifiche anche declinando la richiesta.

3.2.1 Test Cases

Test Case 1:

Così come per TippyTipper, l'applicazione viene testata tramite tecniche di Capture and Replay, cercando di coprire la maggior parte delle funzionalità. In un primo caso di test si vuole aggiungere un nuovo evento: occorre verificare l'esistenza del giorno sulla vista e creare un nuovo evento cliccandolo. Si inserisce il nome dell'avvenimento nel campo caratterizzato dalla scritta di default "Title", infine si verifica che il campo sia stato effettivamente modificato. Nella fase di capture vengono riconosciute tutte le interazioni; in particolare Espresso non registra la pressione sulla tastiera virtuale di tutte le lettere che compongono la frase, ma la frase intera; il comando registrato è quindi *Type into MyEditText with ID event_location: Compleanno Marco*. La fase di replay avviene correttamente, tuttavia è possibile dover aspettare molto tra un'azione e l'altra a causa di alcuni delay registrati nel log; questi non sono associati ad azioni non interpretate da Espresso, ma sono casuali, ed il tempo di attesa raddoppia di volta in volta; ad esempio è capitato che il primo sia dato da una sleep di 40000 millisecondi, il secondo da una di 80000, e così via. Per questo motivo si è scelto di modificare parte del codice da testare, eliminando le sleep o riducendone i millisecondi. Un bug che può verificarsi è quello di accedere più volte al campo da modificare, riuscendo a scrivere la frase solamente in più

tentativi; di fatto l'interazione utente con l'applicazione è quella di cliccare una sola volta l'area caratterizzata da messaggio di default "Title" e modificarlo in "Compleanno", mentre nel log è registrato un primo tentativo di modifica con testo "Comple" ed un secondo "Compleanno". Questo può essere dovuto alle prestazioni dell'emulatore ed al fatto che esso lagghi durante la scrittura.

Test Case 2:

In un secondo caso di test si vuole modificare l'orario di un evento: prima di tutto bisogna scegliere se l'evento sia relativo all'intera giornata oppure ad una particolare fascia oraraia; nel secondo caso bisogna modificare i valori dell'orario relativo a inizio e fine evento, cliccando sui numeri riportati a destra e spostando manualmente le lancette di un orologio virtuale. Quest'ultima azione non viene riconosciuta dallo strumento di capture, motivo per cui il replay termina con esito negativo nel caso vengano effettuate delle asserzioni sull'orario riportato dopo la modifica. In particolare le azioni specifiche del caso di test sono: asserire che esista il campo caratterizzato dal testo 6:00 PM (ora di default), cliccare sul numero e trascinare le lancette, asserire che il campo sia attualmente caratterizzato dal nuovo orario (10:00 PM ad esempio). Nella fase di replay l'azione di modifica non viene riprodotta e di conseguenza l'ultima asserzione fatta risulta errata, facendo terminare il test negativamente. Un'ulteriore osservazione da fare riguardo l'orario è che l'applicazione vuole venire incontro all'utente, impostando come orario di defaul l'ora successiva a quella attuale (in modo da non doverla modificare per eventi quasi immediati). Di conseguenza è possibile che ci siano delle variazioni sull'interfaccia nella fase di replay rispetto a quella di record; ad esempio la fase di record viene effettuata alle 17:05, l'ora di default è 6:00 PM, ed avviando il caso di test in orario differente, 19:30, non verrà riconosciuto il campo in cui modificare l'orario, perchè contrassegnato dal testo 8:00 PM. Per far si che nella fase di replay venga individuato, bisogna cambiare le impostazioni dell'emulatore ed impostare un'ora adatta affinchè quella di default dell'applicazione coincida con quella della fase di capture.

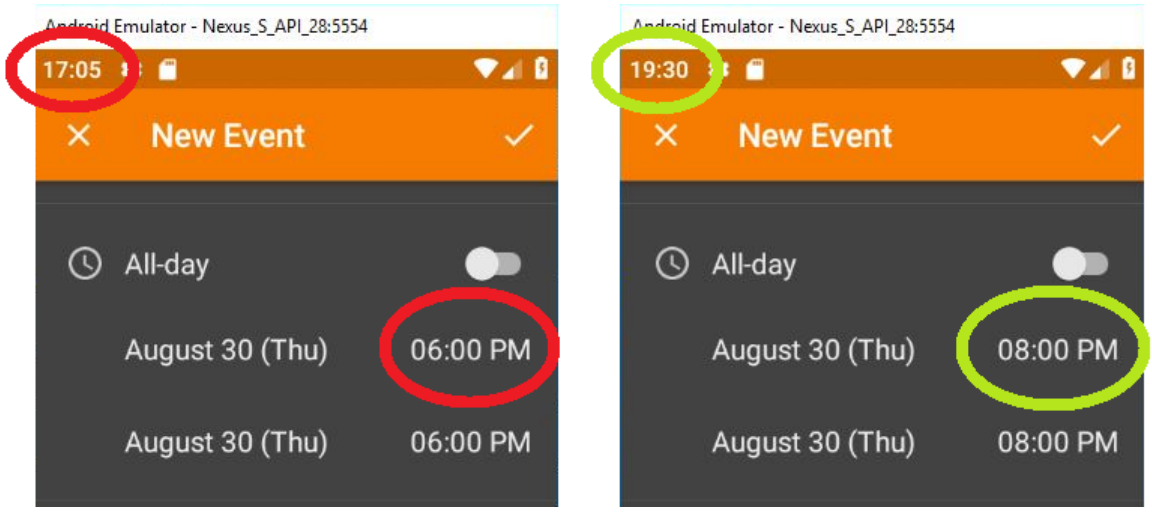


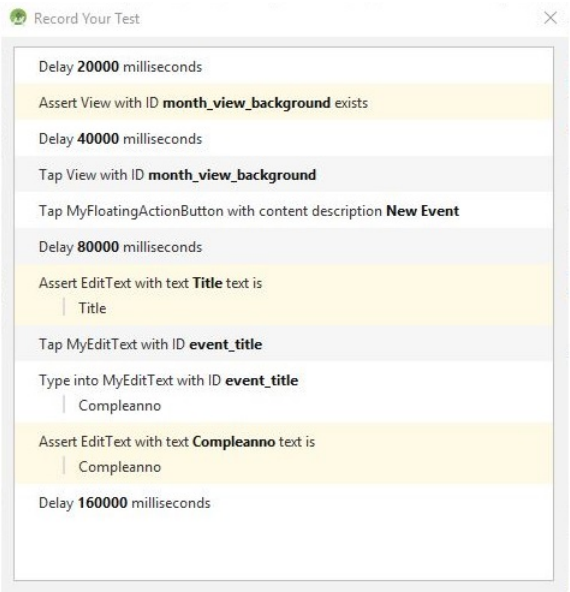
Figura 3.2.2: l'ora di default varia in base all'ora dell'emulatore.

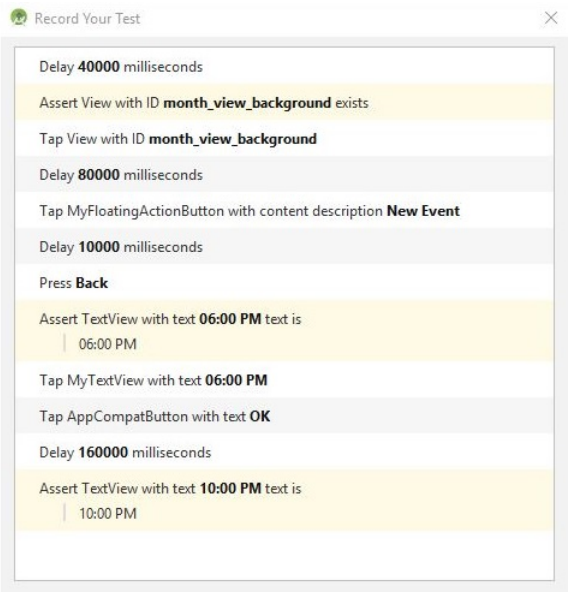
Test Case 3:

Il terzo caso di test mira a verificare il riconoscimento di altre azioni complesse: nel momento in cui si vuole personalizzare un'etichetta per distinguere tra loro più eventi di diverso tipo e visualizzarla in modo efficace sull'interfaccia, bisonga creare un nuovo tipo di evento, scegliere un nome ed associare un colore. Quest'ultimo può essere scelto manualmente da una striscia verticale, regolandone poi l'intensità e la saturazione da una griglia che ne riporta le varie sfumature; ad ogni colore è associato un numero. Per fare ciò bisogna cliccare sul tasto *More Options* in alto a destra, selezionare la voce *Settings* e successivamente *Manage event types*, qui è possibile cambiare nome e colore; se quest'ultimo viene scelto manualmente, spostando il cursore tra le varie sfumature, si ha lo stesso problema riscontrato nel caso di test precedente: sull'interfaccia vengono effettivamente modificati colore e numero ad esso associato, ma l'azione non è rilevata dallo strumento di capture e l'asserzione sul nuovo colore sarà errata in fase di replay. E' possibile arrivare ad un risultato positivo modificando direttamente da tastiera il numero associato al determinato colore; questo tipo di azione (cliccare in un campo e modificarne il testo), utilizzata anche per gli altri casi d'uso, viene interpretata correttamente da Espresso e di conseguenza anche registrata nel log. Il secondo approccio utilizza interazioni utente più

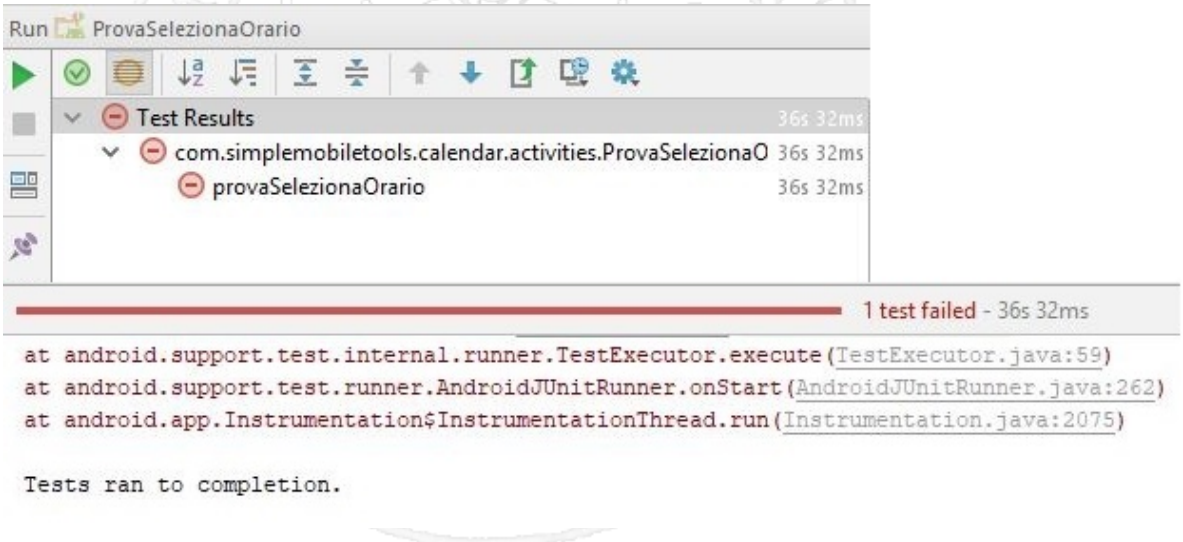
semplici e porta allo stesso risultato, in quanto vi è una relazione biunivoca tra numero e colore.

Di seguito sono riportati graficamente i problemi evidenziati nei vari casi di test:





La figura mostra quanto accaduto nel secondo test case: la prima asserzione indica 06:00 PM, mentre la seconda effettuata sullo stesso oggetto 10:00 PM, senza che sia stata registrata alcuna azione di modifica di testo. Nella fase di replay questo comporta la terminazione negativa del test case con questo messaggio di errore:



Capitolo 4

Code generation settings

Negli scorsi capitoli si è trattato di come Espresso generi automaticamente il codice a partire dalle interazioni utente, analizzando poi i metodi utilizzati e gli oggetti della GUI. La generazione del codice è regolata dai valori assunti da alcune impostazioni, a cui è possibile accedere tramite la sequenza di passi: File -> Settings -> Build, Execution, Deployment -> Espresso Test Recorder.

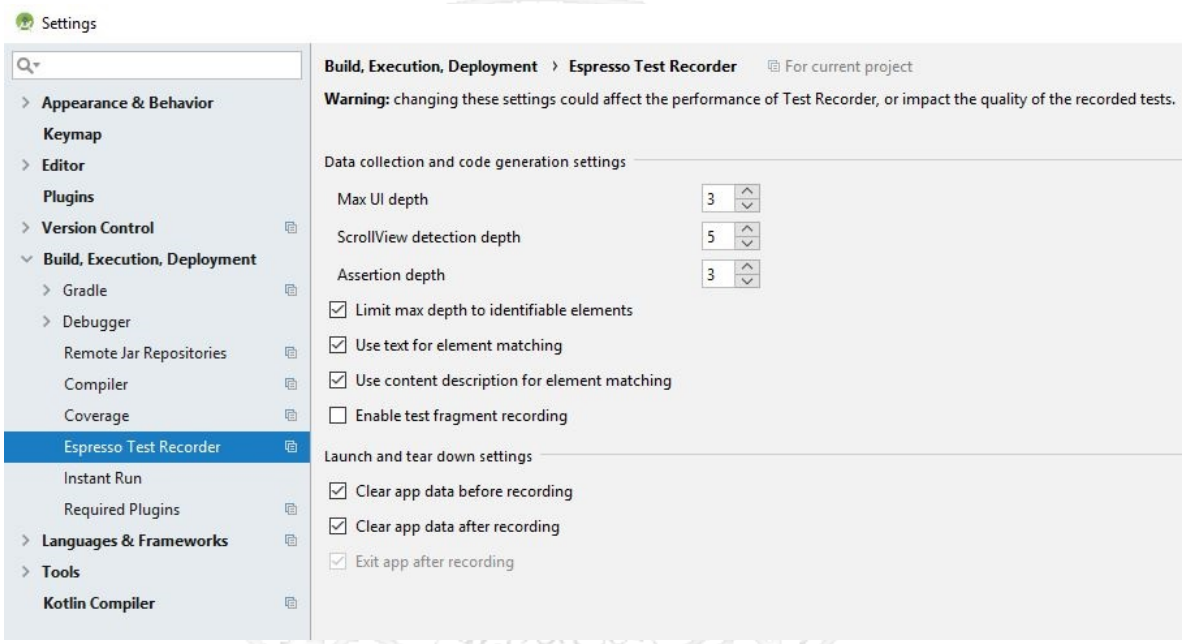


Figura 4.0.1: Data collection and code generation settings.

4.0.1 Max UI Depth

Questo valore indica la "precisione" con la quale un determinato widget è identificato: maggiore è il numero e maggiore è la probabilità che l'oggetto sia definito univocamente. Di conseguenza, al crescere del valore migliora anche la probabilità che il replay vada a buon fine; infatti, un test poco preciso, con bassa profondità, potrebbe incorrere nell'impossibilità di riconoscere il widget tra i vari elementi dell'interfaccia, portando all'interruzione del test. E' possibile notare come questa impostazione influisca sugli attributi che caratterizzano un oggetto, ad esempio, il codice generato con valore valore *Max UI Depth* di default, in seguito alla modifica dell'impostazione *Round Type* in Tippy Tipper, sarà:

```
public void provaDepth3() {
    ViewInteraction button = onView(
            allOf(withId(R.id.Settings), withText("Settings"),
                    childAtPosition(
                            childAtPosition(
                                    withClassName(is( value: "android.widget.LinearLayout")),
                                     position: 3),
                             position: 0),
                    isDisplayed()));
    button.perform(click());


    DataInteraction linearLayout = onData(anything())
            .inAdapterView(allOf(withId(android.R.id.list),
                    childAtPosition(
                            withClassName(is( value: "android.widget.LinearLayout")),
                             position: 0)))
            .atPosition(9);
    linearLayout.perform(click());


    DataInteraction checkedTextView = onData(anything())
            .inAdapterView(allOf(withClassName(is( value: "com.android.internal.app.AlertController$RecycleListView")),
                    childAtPosition(
                            withClassName(is( value: "android.widget.LinearLayout")),
                             position: 0)))
            .atPosition(0);
    checkedTextView.perform(click());
}
```

```
private static Matcher<View> childAtPosition(
        final Matcher<View> parentMatcher, final int position) {

    return new TypeSafeMatcher<View>() {
        @Override
        public void describeTo(Description description) {
            description.appendText("Child at position " + position + " in parent ");
            parentMatcher.describeTo(description);
        }


        @Override
        public boolean matchesSafely(View view) {
            ViewParent parent = view.getParent();
            return parent instanceof ViewGroup && parentMatcher.matches(parent)
                    && view.equals(((ViewGroup) parent).getChildAt(position));
        }
    };
}
```

Figura 4.0.2: La sequenza di azioni registrata è: Settings -> Round Type -> Round Tip

Modificando il valore della profondità, ad esempio impostandola pari ad uno, si avrà un codice più semplice, in cui gli oggetti sono sono definiti attraverso meno attributi. Di conseguenza non vi è necessità di definire il metodo childAtPosition. Tuttavia un test di questo tipo è poco preciso, e nel momento in cui deve inidividuare l'opzione da spuntare *Round Tip* termina con esito negativo.

```
public void provaDepth1() {
    ViewInteraction button = onView(
            allOf(withId(R.id.Settings), withText("Settings"), isDisplayed()));
    button.perform(click());

    ViewInteraction linearLayout = onView(
            allOf(withClassName(is( value: "android.widget.LinearLayout")), isDisplayed()));
    linearLayout.perform(click());

    ViewInteraction checkedTextView = onView(
            allOf(withId(android.R.id.text1), withText("Round Tip"), isDisplayed()));
    checkedTextView.perform(click());

}
```

Figura 4.0.3: Codice ottenuto con Max UI Depth = 1.

4.0.2 Clear app data before/after recording:

Queste impostazioni sono attive di default; disattivandole, c'è la possibilità che il replay fallisca, in quanto i campi modificati potrebbero essere contraddistinti da nuove stringhe di caratteri e non essere più riconosciuti. Ad esempio nel caso in cui in Simple Calendar si voglia modificare l'evento "Compleanno Marco" in "Compleanno Fabio", le prime azioni che coinvolgono elementi della GUI, individuati unicamente da Id, saranno replicate in maniera fedele in fase di replay (*Tap View with ID month_view_background, Tap FrameLayout with ID event_item_frame*); nel momento in cui si accede al campo relativo al titolo dell'evento, nel log viene registrata l'interazione *Tap MyEditText with text Compleanno Marco* e successivamente *Type into MyEditText with text Compleanno Marco: Compleanno Fabio*, essendo contraddistinto sia da Id che da testo. Quest'azione non sarà replicabile in fase di replay in quanto il campo avrà un altro titolo e non verrà individuato sull'interfaccia.

4.0.3 Use Text for element matching:

Modificando questa impostazione abilitata di default, è possibile risolvere il problema precedente. Disabilitandola infatti, l'oggetto in questione nel codice generato avrà come unico attributo l'Id. Una scelta di questo tipo permette di risolvere anche possibili problemi dovuti alla lingua dell'applicazione, ad esempio in un'applicazione che supporta più lingue, uno stesso tasto potrebbe chiamarsi "Save" in inglese ed in italiano "Salva".

```
ViewInteraction myEditText = onView(
        allOf(withId(R.id.event_title), withText("Compleanno Marco"),
                childAtPosition(
                        allOf(withId(R.id.event_holder),
                                childAtPosition(
                                        withId(R.id.event_scrollview),
                                        position: 0)),
                        position: 0)));
myEditText.perform(scrollTo(), click());


ViewInteraction myEditText4 = onView(
        allOf(withId(R.id.event_title), withText("Compleanno Marco"),
                childAtPosition(
                        allOf(withId(R.id.event_holder),
                                childAtPosition(
                                        withId(R.id.event_scrollview),
                                        position: 0)),
                        position: 0)));
myEditText4.perform(scrollTo(), replaceText( stringToBeSet: "Compleanno Fabio"));
```

Figura 4.0.4: l'impostazione è abilitata e l'oggetto ha più attributi

```
ViewInteraction myEditText = onView(
        allOf(withId(R.id.event_title),
                childAtPosition(
                        allOf(withId(R.id.event_holder),
                                childAtPosition(
                                        withId(R.id.event_scrollview),
                                        position: 0)),
                        position: 0)));
myEditText.perform(scrollTo(), click());


ViewInteraction myEditText2 = onView(
        allOf(withId(R.id.event_title),
                childAtPosition(
                        allOf(withId(R.id.event_holder),
                                childAtPosition(
                                        withId(R.id.event_scrollview),
                                        position: 0)),
                        position: 0)));
myEditText2.perform(scrollTo(), replaceText( stringToBeSet: "Compleanno Fabio"));
```

Figura 4.0.5: l'impostazione è disabilitata e l'oggetto non è identificato da testo

Capitolo 5

Espresso Test Recorder - Pro e Contro

In questo capitolo verranno riassunti e schematizzati tutti i punti di forza del tool ed i problemi riscontrati durante il suo utilizzo. Molti sono stati precedentemente analizzati o semplicemente citati; l'intento è quello di elencarli per avere una visione più compatta delle funzionalità offerte, ed un confronto più immediato tra i pro e i contro.

5.1 Pro

- Consente di scrivere casi di test UI automaticamente; in alcuni casi questo approccio è fondamentale, così come anticipato nell'introduzione e poi approfondito nel capitolo 2.
- Permette la scrittura di test cases in maniera veloce, semplice ed interattiva, non dovendo necessariamente studiare il framework.
- Permette di registrare sia interazioni che asserzioni, permettendo di controllare lo stato di determinati widget; ciò rende il test più affidabile.
- Supporta asserzioni multiple.

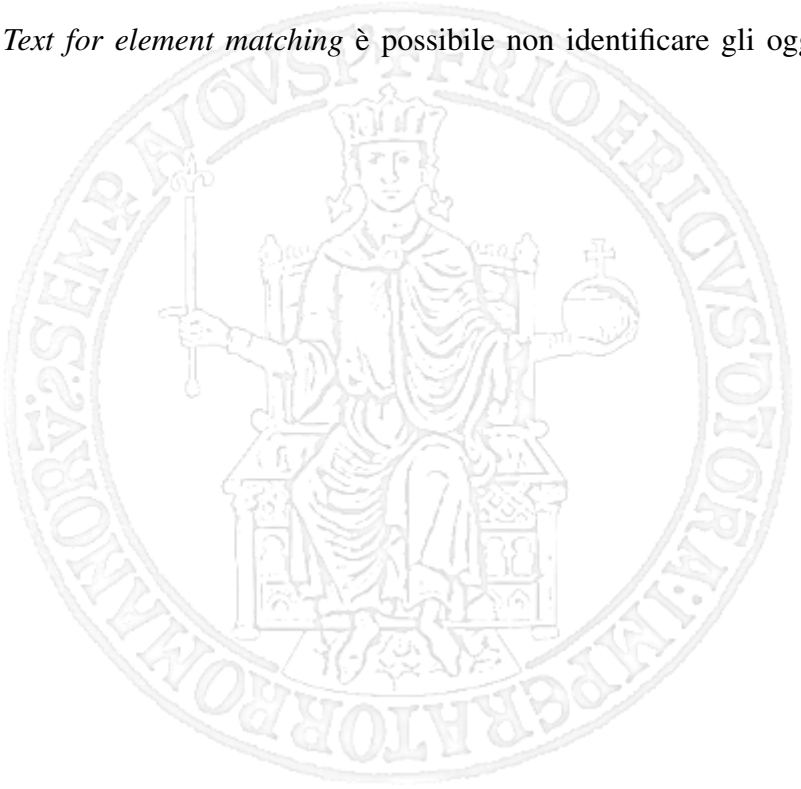
5.2 Contro

- E' possibile che non siano riconosciuti tutti i widget di una stessa interfaccia, stessa cosa per alcune azioni eseguite su di essi (un esempio è lo scorrimento della barra della percentuale in TippyTipper).
- Non supporta il recording di WebViews Interactions.
- Le asserzioni disponibili sono poche e comuni, tra cui: *Exists,* o dualmente *Does not exist*, infine *Text is* per textView e sottoclassi. Nel caso in cui ci fosse bisogno di asserzioni più complicate, esse devono essere introdotte manualmente.
- Le asserzioni non sono riferibili a determinate notifiche, ad esempio i messaggi di tipo Toast. "Il Toast è la forma di notifica più immediata, si tratta di una piccola finestra rettangolare nera che appare nella parte bassa del display contenente un messaggio con il testo bianco. La sua visibilità dura poco, qualche secondo, e le sue apparizioni improvvise dal basso gli hanno donato questo nome che richiama letteralmente il modo in cui il pane salta fuori dai tostapane". [6]
- Nel caso di operazioni asincrone o animazioni, Espresso, non sapendo quanto attendere, aggiungerà delle *Thread.sleep( )* ed il seguente commento "*Added a sleep statement to match the app's execution delay. The recommended way to handle such scenarios is to use Espresso idling resources: https://google.github.io/android-testing-support-library/docs/espresso/idling-resource/index.html*". Tuttavia questo approccio non risolve il problema, in quanto nel test si resta in attesa di eventi UI e/o network call, quindi operazioni asincrone, e come dice il commento occorre risolvere utilizzando le Idling Resources manualmente. [8]
- Non ottimizza la dimensione del codice, questo può risultare ridondante; ad esempio: cliccando tre volte il pulsante zero in "TippyTipper", viene creato tre volte lo stesso oggetto; sarebbe più opportuno invece creare l'oggetto una sola volta e chiamare il metodo quando serve. Ancora, quando il valore della Max UI Depth viene incrementato a favore di quegli oggetti difficilmente identificabili, anche tutti gli altri oggetti verranno caratterizzati da molti attributi, spesso più attributi di quanti

servano realmente.

- E' possibile che il replay di un caso di test fallisca se eseguito su un dispositivo che abbia impostata una lingua diversa, questo perchè uno dei metodi proposti da Espresso per l'individuazione univoca di un widget è quello di riconoscere in funzione del testo. Tuttavia esso potrebbe cambiare a seconda della lingua, rendendo i test non più funzionanti.

Per ottimizzare il codice è possibile affiancare alla sua generazione automatica un processo di refactoring. "Il refactoring è una tecnica utilizzata per modificare la struttura interna di porzioni di codice senza modificarne il comportamento esterno" [9]; un processo di questo tipo, quindi, prevede una modifica (sia essa manuale, semiautomatica, o automatica nei casi migiori) che lasci inalterato il comportamento del codice e che porti dei vantaggi, quali ad esempio un miglior utilizzo delle risorse di memoria, migliore leggibilità, complessità minore; o ancora che favorisca la manutenibilità e la riusabilità. Questa potrebbe essere una soluzione anche all'ultimo problema presentato, ricordando che un'ulteriore alternativa è già stata presentata nel capitolo 4: modificando l'impostazione *Use Text for element matching* è possibile non identificare gli oggetti in base al testo.



Conclusioni e sviluppi futuri

Ringraziamenti

Giunto alla conclusione di questo lavoro, desidero ringraziare il professor Tramontana, relatore di questa tesi di laurea, per essere stato disponibile e preciso durante l'intero periodo di stesura dell'elaborato.

Bibliografia

[1] Carli M., “Android: guida per lo sviluppatore”, Apogeo, 2010

[2] Tramontana P., “Android”, slides di Ingegneria del Software 2

[3] Tramontana P., “Android Testing”, slides di Ingegneria del Software 2

[4] Tramontana P., “User Session Based Testing”, slides di Ingegneria del Software 2

[5] Maggi G., “Introduzione: perchè Android”,
sito: http://www.html.it/pag/19496/introduzione-perch-android/

[6] Maggi G., “Notifiche: Toast e Dialog”,
sito: http://www.html.it/pag/48933/notifiche-toast-e-dialog/

[7] Android Developers, “Create UI tests with Espresso Test Recorder”,
sito: https://developer.android.com/studio/test/espresso-test-recorder

[8] Lannunziato P., “Android Testing: Espresso Test Recorder”,
sito: https://medium.com/@Pask23/android-testing-espresso-test-recorder-e33c1a219fad

[9] Wikipedia, “Refactoring”, sito: https://it.wikipedia.org/wiki/Refactoring