Politecnico di Torino

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Gamification Applicata al GUI Testing di Applicazioni Mobile

Relatore:
Prof. Luca Ardito

Candidato:
Fulcini Tommaso

Co-relatore:
Dott. Riccardo Coppola

Anno Accademico 2020/21
Torino

Sommario

***Contesto*:** Il GUI Testing è una disciplina del Software Testing basata sull'interazione grafica tra il tester e l'applicazione target. Nonostante tale pratica sia un valido mezzo di validazione dell'app sviluppata, spesso i team di sviluppo tendono a trascurarla a causa dei costi elevati derivanti da un approccio manuale e dalla fragilità dei test risultanti da metodi puramente automatizzati. Un approccio più recente è fornito dall'Augmented Testing, che permette di combinare l'esaustività dei test manuali con la rapidità di esecuzione e il risparmio di tempo tipici dell'automatizzazione.

***Obiettivo*:** Questa tesi propone l'applicazione di alcuni concetti di Gamification al GUI Testing, ossia l'inserimento di elementi e meccanismi tipici del game design all'interno di contesti non ludici, in questo caso nell'esecuzione delle sessioni di testing, al fine di migliorare la qualità delle test suite risultanti dalle sessioni di testing e di stimolare l'aspetto emotivo del tester.

***Metodo*:** La tesi si concentra sulla teorizzazione di cinque elementi ludici e sulla loro implementazione in un plugin Java per un prototipo di software di Augmented Testing: Scout.

***Risultati*:** Un campione ristretto composto da studenti e lavoratori, laureati (o laureandi) in Informatica o Ingegneria Informatica ha utilizzato il prototipo di Scout *as is* e con il plugin di Gamification attivo; è stato notato un generale miglioramento nella soddisfazione degli individui dal punto di vista emotivo e nelle performance, con un incremento medio del 43% nella coverage globale raggiunta.

***Conclusioni*:** L'implementazione realizzata ha mostrato come la consapevolezza delle performance e la soddisfazione siano migliorate a seguito dell'uso della Gamification, confermando le potenzialità dell'applicazione di tale tecnica.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Background e Stato dell'Arte

1.1 Introduzione

Nello sviluppo di un software un'attività fondamentale è quella del testing: questa fase può incidere anche per oltre il 40% del costo totale. Nonostante l'importanza di tale attività, spesso le aziende finiscono per lesinare, dedicando un quantitativo di tempo e di risorse molto inferiore rispetto a quello che normalmente dovrebbe richiedere. Svolgere la mansione di testing inoltre è considerata una pratica noiosa e poco stimolante.

Testare in maniera inadeguata il software in produzione espone a rischi molto elevati: è molto frequente che, in seguito a queste lacune, il software prodotto possa essere affetto da problemi di sicurezza, comportamenti indesiderati e disservizi che vanno a ledere l'esperienza che l'utente finale si ritrova ad avere. Risolvere i problemi presenti dopo il rilascio del software è inoltre molto più costoso che farlo in seguito ad una accurata fase di testing atta a rivelare i malfunzionamenti nel codice.

In questo studio ci si concentrerà sul GUI Testing: un tipo di software testing che si focalizza sugli aspetti legati all'interfaccia grafica facendo controlli sulle schermate (i bottoni, le aree di testo, le icone e tutti gli altri elementi visivi). Questa categoria di test è particolarmente importante nel contesto delle applicazioni web e delle app mobile in cui le interfacce grafiche sono in continua evoluzione e possono cambiare release dopo release (con piccole modifiche o addirittura stravolgimenti completi).

Questo lavoro, in particolare, tratterà dello sviluppo di un plugin che introduca delle meccaniche di Gamification ad un software usato per testare l'interfaccia

grafica, al fine di stimolare i tester nelle loro mansioni. Di seguito verrà analizzato il tema della Gamification da diverse prospettive e verranno mostrati gli studi precedenti che sono stati analizzati e sulla base dei quali si è costruito il plugin.

1.2 Gamification

La **Gamification** (o *ludicizzazione*) consiste nell'uso degli elementi, della filosofia e delle meccaniche tipiche del game design in contesti non ludici (Deterding et al., 2011). Esistono diversi approcci per applicare gli elementi tipici dei giochi al fine di supportare obiettivi diversi dal consueto intrattenimento: per esempio, in diversi contesti vengono utilizzati i cosiddetti *serious games*, ossia giochi veri e propri che hanno finalità tipicamente educative. La Gamification, però, si differenzia da questi ultimi in quanto un sistema ludicizzato non è un gioco in tutto e per tutto, ma uno strumento che ha altre finalità e che integra al suo interno delle meccaniche di gioco per cercare di ottenere determinati benefici. Un sistema ludicizzato, quindi, si compone integrando un ambiente di Gamification nel contesto dell'attività principale.

L'idea alla base del concetto, quindi, è quella di sfruttare tutte quelle caratteristiche che rendono i giochi attraenti e divertenti (al punto tale da creare dipendenza in certi casi), per migliorare l'esperienza di un certo target di utenti in ambienti non ludici come il posto di lavoro, la scuola o l'università, ossia tutti contesti dove lo scopo principale non è il divertimento.

1.2.1 Teorie sull'applicazione della Gamification

Molti studi sulla Gamification si concentrano sui risultati ottenuti dall'applicazione di tali tecniche nei vari ambiti e sui benefici che esse possono apportare, mentre solo una piccola parte affronta il problema dal punto di vista del design di un sistema ludicizzato. Nonostante tutti i benefici citati in precedenza, infatti, le applicazioni ludicizzate non sono semplici da implementare, poiché si deve tenere conto di molti fattori per evitare di realizzare un sistema scadente e fallimentare.

Un modo per identificare e tentare di risolvere alcuni eventuali problemi fin dalla fase di progetto di un sistema ludicizzato consiste nella creazione (o nell'utilizzo) di un framework metodologico per l'integrazione della Gamification nei processi di

sviluppo di software, come quello proposto in (Herranz et al., 2014) e (Herranz et al., 2013) e rappresentato in Figura 1.1.

Figura 1.1. Fasi del framework.

Il framework si suddivide di 7 fasi:

1. *Viability*: questa fase valuta la fattibilità dell'implementazione della Gamification nell'organizzazione in esame;
2. *Business Objectives*: si fa un'ulteriore valutazione riguardo alla possibilità di implementazione della Gamification sulla base degli obiettivi del progetto;
3. *Player Objectives and Motivations*: si esplorano le motivazioni dei vari gruppi professionali che lavoreranno al progetto;
4. *Activities to Enhance*: si identificano le attività del processo migliorabili grazie alla Gamification e si discute sulle strategie da adottare per farlo;
5. *Gamification Proposal*: il fulcro del framework, la fase in cui si propone un'effettiva implementazione del sistema ludicizzato;
6. *Implementation*: effettiva realizzazione e integrazione della proposta della fase 5;

7. *Assessment*: si analizzano i risultati ottenuti e gli obiettivi raggiunti.

Un altro framework per la ludicizzazione dell'Ingegneria del Software è stato presentato da (Dal Sasso et al., 2017) ed è rappresentato in Tabella 1.1. Esso si basa sul concetto di **Attività**, a sua volta composta dai concetti di **Analisi**, **Implementazione** e **Testing**. Ogni attività riguarda uno specifico tipo di utente (*ruolo*) presente nei sistemi ludicizzati, tra cui troviamo:

- *Observer*: agiscono in modalità di sola-lettura;
- *Writer*: interagiscono modificando il contenuto già presente;
- *Solver*: realizzano gli obiettivi del sistema ludicizzato.

Normalmente le persone che interagiscono col sistema passano dinamicamente da un ruolo all'altro.

Come detto in precedenza, il concetto fondamentale del framework è l'**Attività**, che è composta da un ID formato dall'iniziale del ruolo concatenata con un numero incrementale, una breve descrizione e una lista dei più pertinenti *elementi costitutivi della Gamification* (analizzati in seguito). Ogni attività viene a sua volta strutturata in tre fasi:

1. **Analisi**: ogni attività all'interno dell'ambiente ludicizzato deve basarsi su un *fondamento logico* facilmente comprensibile da collegare agli obiettivi globali dell'ambiente, oltre che su un *obiettivo emotivo*, che rappresenta i benefici che si vogliono ottenere nelle persone che interagiscono con il sistema.
2. **Implementazione**: per implementare un'attività bisogna innanzitutto avere una chiara idea degli *attori* in gioco, oltre che delle *dinamiche* in cui verranno coinvolti, che rappresentano sostanzialmente le tattiche per coinvolgere le persone in una certa attività. Tutto ciò viene istanziato utilizzando componenti tipici del game design chiamati *meta*, come livelli, vite e classifiche. Infine, è necessario riflettere sui *rischi* che possono sorgere dalla struttura ludica scelta (ad esempio problemi algoritmici, comportamenti indesiderati, requisiti dell'hardware ecc.).
3. **Testing**: fase che consiste nel capire quali siano i *target* dell'attività di testing, quale *metodologia* possa essere utilizzata per eseguire i test e, infine,

quali siano i *risultati attesi*, da confrontare con quelli *reali*, per agevolare un approccio iterativo di sviluppo del sistema ludicizzato.

Attività	
Ruolo & ID	
Descrizione	
Elementi Costitutivi	
Analisi	Fondamento Logico Obiettivo Emotivo
Implementazione	Attori Dinamiche Meta Rischi
Testing	Target Metodologia Risultati Attesi Risultati Reali

Tabella 1.1. Template di un'Attività

Un'attività dipende da uno o più **elementi costitutivi** della Gamification. Ne esistono dieci tipi:

1. *Portale*: consiste in una ricompensa (ad esempio un badge) di benvenuto, assegnata ad un utente non appena attraversa i confini della piattaforma, registrandosi e fornendo le informazioni necessarie per farsi identificare all'interno della community.

2. *Produzione*: elemento che ha lo scopo di rendere immediatamente produttivo un utente sulla piattaforma, per evitare che ci sia un calo di interesse. Si può suddividere in tre sotto-elementi:

   - *Simbiosi*: si esegue un'attività che aiuta direttamente o indirettamente un altro utente;
   - *Narciso*: si esegue un'azione per migliorare la propria posizione all'interno della community, così da aiutare l'utente a capirne i meccanismi;

- *Alveare*: si deve proporre un'idea per migliorare la piattaforma e la vita della community.

3. *Coraggio*: è importante premiare in maniera equa gli utenti che riescono ad eseguire determinati task, specialmente i più complicati. Aumentare la fiducia in sé stessi degli utenti potrebbe stimolarli a voler provare task più difficili, il cui completamento potrebbe accrescere le loro competenze e quindi renderli in grado di affrontare attività progressivamente più complicate.

4. *Mischia*: questo elemento fa riferimento alla mischia del rugby per porre enfasi sull'importanza della forza del gruppo. Collaborare, condividere risorse utili, competere e socializzare con altri membri della community è intrinsecamente motivante, quindi il sistema deve premiare e promuovere il lavoro di squadra e la competizione.

5. *Camaleonte*: elemento che fa riferimento alla dinamicità con cui devono mutare gli achievement proposti dal sistema. Supponendo che l'utente, utilizzando il sistema, acquisisca sempre più competenze che lo spingeranno a cercare sfide sempre nuove (*Coraggio*), l'ambiente dovrebbe reagire a questi cambiamenti introducendo nuovi achievement e rilasciando ricompense ad-hoc, che entreranno poi a far parte della libreria di sistema così da poter essere ottenuti anche da altri utenti. Allo stesso modo, achievement che non vengono raggiunti da nessun utente per troppo tempo dovrebbero essere eliminati dalla libreria per via della loro probabile impraticabilità.

6. *Fulmine*: una volta diventato esperto, l'utente potrebbe annoiarsi e si potrebbe assistere ad un calo della sua produttività. È in questa situazione che l'ambiente dovrebbe "colpire l'utente come un fulmine", dirigendolo verso una nuova sfida particolarmente motivante con premi personalizzati.

7. *Phasing*: gli utenti nel mondo virtuale possono compiere delle azioni che hanno un impatto permanente sull'ambiente circostante. Questo elemento suggerisce di mutare la piattaforma in relazione alla progressione della competenza di ciascun utente. Di conseguenza, due utenti in diverse fasi della propria progressione vedranno rappresentazioni differenti dello stesso scenario e potranno interagire con esso in modi diversi.

8. *Abbellimento*: è importante che l'aspetto degli avatar degli utenti cambi nel tempo, diventando sempre più attraente man mano che si progredisce nell'ambiente virtuale. Al contrario, in caso di inattività, deve degradarsi sempre di più col passare del tempo.

9. *Champagne*: dato che gli achievement virtuali sono importanti per gli utenti, bisogna permettere loro di celebrare i propri successi virtuali anche nel mondo reale, per esempio permettendo di inserire all'interno del proprio CV, un link diretto al proprio profilo virtuale.

10. *Ascensione*: normalmente i giochi hanno una fine, ma nel contesto di un processo produttivo è importante non porre un limite del genere alle attività per evitare che alcuni utenti diventino inattivi. È bene però premiare nel mondo reale gli utenti che hanno ottenuto risultati particolarmente positivi all'interno dell'ambiente virtuale.

Oltre ai framework appena descritti, sono stati realizzati numerosi studi considerabili come punto di partenza per il design di un sistema ludicizzato. Tra i più significativi, troviamo un'analisi effettuate dal punto di vista della figura del project manager volte a studiare come applicare le tecniche di Gamification ad un progetto.

Nello studio di (Platonova & Bērziša, 2017) viene mostrato come i benefici della Gamification (aumento della motivazione, riduzione dello stress e intrattenimento durante un'attività lavorativa) siano alcuni degli obiettivi ricercati da un project manager per il proprio team di lavoro.

Secondo le autrici, gli elementi di Gamification più comuni e di maggior successo sono basati su punteggi e classifiche: l'efficacia di tali elementi è dovuta al fatto che viene dato un feedback immediato al giocatore; l'assegnazione dei punteggi ai task da completare deve seguire un criterio proporzionale che può tuttavia anche condurre al fenomeno della *pointsification*. Questo comportamento indesiderato verrà discusso ampiamente in seguito.

La classifica è un altro elemento importante in quanto inserisce una componente di rivalità e di sfida tra le persone coinvolte nell'ambiente ludicizzato; i badge e gli achievements invece sono meccaniche meno utilizzate secondo le autrici poiché meno motivanti in quanto serve molta pazienza per ottenerli.

Secondo (Dal Sasso et al., 2017) quando ci si approccia al design di un ambiente ludicizzato si deve considerare i seguenti aspetti ponendosi le seguenti domande:

- *Determinazione*: da dove scaturisce la motivazione per incoraggiare un certo comportamento?
- *Scelte significative*: le attività scelte per la ludicizzazione sono abbastanza significative?
- *Struttura*: si può spiegare con un algoritmo il comportamento desiderato?
- *Potenziale conflitto*: Il gioco evita la tensione con un'altra struttura emotiva (gioia, competitività, ecc.)?

Inoltre, la Gamification deve avere lo scopo di espandere le interazioni già esistenti piuttosto che quello di rendere divertente un compito noioso: questo perché si rischia di ottenere un effetto contrario (repulsione per una Gamification imposta o fenomeno della *pointsification*).

Infine, si propone una serie di indici per valutare l'efficacia dell'impatto del layer di Gamification, tre di questi dal punto di vista tecnico e due dal punto di vista emotivo: *success metrics, analytics, conflitti, jen ratio, questionari.*

- *Success Metrics*: quantità definite per ogni attività; devono essere specifiche per ogni piattaforma in modo che si riferiscano a quantità significative e facili da misurare.
- *Analytics*: sono metriche che misurano l'interazione degli utenti con la piattaforma (dati come il *Daily Active User* servono per contare gli utenti unici di un certo giorno, mentre il rapporto *Daily/Monthly Active User* fornisce un trend sull'uso del tool in un certo momento).
- *Conflitti*: sono i possibili errori che sorgono dal momento dell'introduzione del layer di Gamification. Tali errori devono essere al più presto identificati e risolti, altrimenti, va rimosso il layer stesso.
- *Jen Ratio*: valuta quanto sia positivo l'atteggiamento degli utenti misurando le interazioni positive e quelle negative (rapporto tra le interazioni positive totali tra gli utenti rispetto alle interazioni negative totali in un dato periodo

di tempo e contesto). Il risultato è compreso tra 0 e 1: più è prossimo a 1, migliore è il benessere percepito dalla comunità.

- *Questionari*: vengono somministrati periodicamente a utenti di diversi livelli di competenze ponendo domande non solo tecniche, riguardanti il funzionamento del sistema, ma anche sullo stato emotivo derivante dall'uso del sistema.

È stato infine evidenziato come un buon indice di predizione per un sistema ludicizzato sia il fatto di poter conoscere anticipatamente se i giocatori, ossia gli utilizzatori del sistema, giochino nel loro tempo libero o no (Mollick & Rothbard, 2014): una persona già abituata avrà meno difficoltà a comprendere le meccaniche di gioco e avrà un interesse maggiore.

1.2.2 Applicazioni nell'Istruzione

Nel campo dell'istruzione, la Gamification viene utilizzata principalmente per aumentare la motivazione, il coinvolgimento e le performance degli studenti durante i corsi. Di seguito verranno presentati alcuni esempi di applicazione delle meccaniche ludiche in ambito accademico.

Nel loro studio, (Anderson P. & R., 2015) descrivono e analizzano un tentativo di ludicizzazione di un corso di data science tramite **Learn2Mine**: un sistema di apprendimento open-source cloud-based che sfrutta la Gamification per formare data scientist in maniera divertente e coinvolgente.

Il sistema è basato su un meccanismo di scomposizione di un problema più complicato in diversi sottoproblemi e step intermedi (facoltativi) proposti in maniera incrementale. L'idea è quella di guidare lo studente nella scrittura del codice richiesto partendo dalla realizzazione delle sue varie parti (spesso indipendenti le une dalle altre, eccezion fatta per lo step finale che dipende da tutti i precedenti).

Learn2Mine sfrutta principalmente barre dell'esperienza, punti, badge e classifiche per stimolare la competizione fra gli studenti. Il sistema fornisce inoltre un feedback immediato sull'attività svolta dallo studente ed elargisce ricompense e achievement in base ai risultati raggiunti. Tutti questi elementi sono integrati in maniera *non intrusiva*, permettendo a chiunque non fosse interessato agli aspetti legati alla Gamification di ignorarli con facilità.

Learn2Mine, inoltre, grazie al meccanismo di feedback immediato, solleva gli insegnanti dall'incarico di dover fornire assistenza continua agli studenti, permettendo loro di avere una panoramica sui progressi raggiunti nell'apprendimento. I docenti hanno inoltre la possibilità di distribuire manualmente delle ricompense agli studenti per i loro risultati parziali o di modificare i voti (e/o i punteggi) che non ritengono coerenti con le linee guida stabilite all'inizio del corso.

Al termine del corso sono state raccolte le valutazioni degli studenti riguardo al sistema tramite un questionario basato su scala Likert e la media delle valutazioni è risultata concorde sull'efficacia del sistema. In particolare, i vantaggi principali sono stati individuati nella possibilità di ritentare più volte di risolvere uno stesso problema e nei feedback immediati forniti dal sistema. Learn2Mine, inoltre, ha aumentato la percentuale di consegna delle soluzioni portandola dal 40% all'80-90% di media nel giro di un solo anno di utilizzo. Secondo gli autori dello studio, Learn2Mine ha portato un ulteriore miglioramento sia nella comprensione che nell'apprendimento degli argomenti proposti.

Gli studenti hanno tuttavia mosso alcune lamentele verso le funzionalità del sistema: alcune riguardavano il fatto che la scomposizione dei problemi in sottoproblemi più piccoli fosse eccessiva, altre il fatto che gli step intermedi fossero tutti indipendenti dagli altri, altre ancora il fatto che gli errori segnalati dal sistema fossero poco chiari.

Un altro tentativo di applicazione della Gamification all'ambito accademico è stato effettuato da (Buckley & Clarke, 2018) che hanno proposto l'erogazione di un corso di *Software Testing* in due modalità, allo scopo di metterle a confronto.

Le modalità di insegnamento confrontate sono: *SEP-CyLE* (Software Engineering and Programming Cyberlearning Environment) che include collaborative learning, social interaction e problem-based learning e *WReSTT-CyLE* (Web-Based Repository of Software Testing Tutorials: a Cyberlearning Environment) che al contrario non include gli elementi di problem-based learning, chat e ratings ed è più simile ad una raccolta di tutorial riguardanti gli argomenti delle varie lezioni.

La Gamification è stata applicata tramite l'attribuzione di punti nei seguenti modi: quando si completa un Digital Learning Object, superando i quiz, interagendo sul forum e completando le attività. È presente anche una parte di interazione sociale realizzata tramite la creazione di un profilo per lo studente, l'invio di messaggi nel forum per la comunicazione e il rating di post e Digital Learning Object;

inoltre troviamo anche una classifica che mostra gli avatar dei 3 studenti che hanno ottenuto più punti.

Uno studio più recente ha provato ad applicare la Gamification in modo differente: invece che focalizzarsi sulla metodologia di erogazione della didattica come nei casi descritti in precedenza, integra le meccaniche ludiche nella valutazione del codice prodotto degli studenti durante il corso (Zsigmond et al., 2020).

Il sistema proposto ha lo scopo di insegnare concetti di programmazione usando come elementi di Gamification i punti e i badge. Viene utilizzata una piattaforma web in cui ogni studente deve consegnare il proprio codice entro determinate scadenze, stabilite anticipatamente per ogni assignment. Il sistema analizza il codice presentato prima in modo statico, controllandone la correttezza sintattica, e poi dinamicamente, ponendolo in esecuzione in una sandbox e ridirigendo input e output in modo controllato: per questo controllo dinamico vengono usati un numero predefinito di test cases dipendenti dall'assignment.

I badge hanno un prerequisito di sblocco noto a tutti gli studenti, che quando soddisfatto, certifica di aver raggiunto certi obiettivi: questo ha lo scopo di suscitare l'interesse e l'impegno dei partecipanti a produrre codice funzionante e ad applicare i concetti appresi. Vi sono due tipi di badge:

- *Di stile*: incoraggiano gli studenti all'adozione di pratiche di programmazione corrette e a cimentarsi nello sviluppo di codice in tal senso;
- *Achievement*: certificano le performance dello studente e forniscono inoltre dei punti bonus validi per l'esame finale, che spingono gli studenti a competere tra loro in quanto alcuni badge possono essere assegnati solo ad un numero limitato di persone.

1.2.3 Applicazioni nell'Ingegneria del Software

La Gamification merita una menzione particolare per quanto riguarda la sua applicazione nel campo dell'Ingegneria del Software, poiché i processi di sviluppo di software sono fortemente human-intensive e questo la rende uno strumento promettente, in grado di aumentare il coinvolgimento quotidiano e la motivazione degli sviluppatori.

Nello studio realizzato da (Passos E.B. & E.W.G., 2011) viene descritto un tentativo di Gamification del processo di sviluppo di un software. L'idea di base nasce non solo dal tentativo di sfruttare i benefici che la ludicizzazione potrebbe potenzialmente apportare (quali maggior divertimento e coinvolgimento degli sviluppatori), ma anche dall'idea di vedere lo sviluppo di un software come un susseguirsi di task (*missioni*) durante i quali si devono acquisire delle competenze per poter superare sempre nuove sfide, ottenendo ricompense o penalizzazioni a seconda del risultato.

Lo studio si concentra molto sul game design, ponendo particolare enfasi sull'organizzazione e la gerarchia dei livelli (disposti in un grafo orientato contenente le sfide) e lo skillset necessario per superarli. Sono inoltre presenti altre caratteristiche:

- L'*esito* di una sfida è modellato con una funzione che ritorna 1 in caso di successo, 0 in caso di fallimento, prendendo in input le variabili che compongono lo stato del gioco al momento della chiamata. Il risultato di una sfida dipende dal risultato ottenuto dal giocatore nelle n sottosfide che compongono quella principale.

- Gli *achievement*, assegnati in due modi: sulla base della *repetition*, in base a quante volte uno stesso tipo di sfida è stato superato e assegnati basandosi su una scala logaritmica che rende i livelli successivi via via più difficili da superare, oppure sulla base del *rate*, la percentuale di successi rispetto ai tentativi in un certo intervallo di tempo, assegnati basandosi su una scala lineare e sul superamento di determinate soglie. Gli achievement vengono distribuiti sulla base di diverse metriche (per esempio la code coverage nel caso del testing) sia individuali che collettive (relative ai progetti).

- *Feedback* immediato, al quale viene data molta importanza, in quanto viene ritenuto un aspetto fondamentale per rendere il giocatore consapevole dei propri progressi.

Andando a mappare tutti questi concetti di game design con le fasi tipiche dell'Ingegneria del Software, le sfide equivalgono di fatto alle release di un software e ognuna di esse si può scomporre in una serie di iterazioni che costituiscono le sottosfide o i task. I nodi foglia del grafo dei livelli sono task immediati.

È importante calibrare attentamente la difficoltà delle sfide per renderle interessanti, evitando che siano troppo facili o troppo difficili così da non incorrere nel fenomeno del *burnout*, ossia la situazione in cui una skill viene padroneggiata ad

un livello tale da renderne noioso l'utilizzo. Un buon modo per evitare il burnout è quello di usare espedienti come lo storytelling, in modo da distrarre il giocatore e mantenerlo interessato al gioco.

Il sistema analizzato nello studio sfrutta elementi tipici dei *Role-Playing Games* (noti anche come *RPG* o *giochi di ruolo*), come punti esperienza (assegnabili anche in numero minore se un task non viene eseguito entro una certa scadenza, per esempio), attributi del personaggio (i cui punti vengono assegnati automaticamente in base alle sfide superate e non scelti dal giocatore) e classi (anch'esse automaticamente attribuite).

Gli autori dello studio hanno notato che l'uso degli achievement non ha soltanto incrementato il coinvolgimento degli sviluppatori, ma ha anche permesso di monitorare e controllare il processo di sviluppo nel suo complesso. Si pone inoltre l'accento sulla possibilità di creare un legame fra uno sviluppatore e il suo avatar nel gioco, cosa che può portarlo a svolgere con più interesse le proprie attività quotidiane. Questo aspetto è legato al fatto che, diversamente da quanto accade in un videogioco, le abilità del personaggio e i suoi progressi sono strettamente legati al programmatore in quanto riflettono i suoi progressi nel mondo reale. Tutti questi aspetti hanno portato gli autori a valutare positivamente l'integrazione del layer di Gamification nel processo di sviluppo.

Un altro esempio di applicazione della Gamification all'Ingegneria del Software, questa volta incentrato sul suo insegnamento piuttosto che su un effettivo processo di sviluppo, viene descritto da (Berkling & Thomas, 2013). Anche in questo caso, il principale motivo dell'adozione di un sistema di ludicizzazione è legato ai benefici che questo tipo di approccio riesce spesso ad apportare in termini di partecipazione e motivazione dei soggetti coinvolti. In particolare, il corso di Ingegneria del Software analizzato nello studio è stato organizzato proprio concentrandosi sulle teorie psicologiche riguardanti la motivazione: secondo diversi studi, infatti, gli esseri umani sono motivati a lavorare su task difficili dal punto di vista cognitivo quando vengono loro garantite autonomia (*autonomy*), uno scopo (*purpose*) e la possibilità di padroneggiare ciò che stanno facendo (*mastery*). Il corso ha tenuto quindi conto di questi 3 aspetti:

- è stato seguito un percorso chiaro con varie possibilità in termini di ordine e velocità di svolgimento delle attività (*autonomy*);

- è stato stabilito un obiettivo chiaro e sperabilmente gratificante (*purpose*);
- gli studenti hanno avuto la possibilità di sostenere più volte alcuni esami durante il corso (*mastery*).

Tutti questi concetti sono stati presi in considerazione nell'integrazione della Gamification, applicata inserendo elementi quali progress bar, classifiche e ricompense. Era inoltre possibile vedere i progressi degli altri studenti e mandar loro messaggi, così da incentivare non solo la competizione ma anche la collaborazione (premiata con ricompense speciali).

L'idea di base è quella di organizzare gli argomenti del corso su una mappa divisa in diverse aree, ognuna rappresentante un diverso topic e ognuna suddivisa in 3 parti. L'obiettivo di uno studente è quello di acquisire un certo insieme di conoscenze completando almeno due parti su tre di una determinata area, così da sbloccare l'accesso al *marketplace* di quell'area e di conseguenza la possibilità di creare un gruppo di colleghi per poter iniziare un progetto. All'interno di questi *marketplace* è possibile pubblicizzare il proprio gruppo o unirsi ad uno di quelli esistenti. È inoltre presente una chat con cui gli studenti possono chiedere aiuto o fornire assistenza ai propri colleghi, con la possibilità di ottenere dei punti bonus.

L'esperimento non ha ottenuto i risultati sperati, ma gli studenti hanno trovato interessanti gli elementi legati alla *task overview*, ossia la possibilità di monitorare i progressi raggiunti nelle proprie attività, e gli elementi legati al *feedback*.

Un'applicazione della Gamification ad un aspetto dell'Ingegneria del Software piuttosto inusuale, ossia la definizione delle specifiche formali, è quella eseguita con **FormalZ** (Prasetya et al., 2019). Da quello che ci è dato sapere, questo è l'unico tentativo di usare tecniche di ludicizzazione per insegnare a livello accademico i concetti relativi a questa fase dello sviluppo del software.

Gli aspetti ludici sono molto ben studiati e integrati: viene raccontata la storia di un hacker che cerca di compromettere una CPU e il giocatore ha come scopo quello di proteggerla dall'attacco. Per attuare delle protezioni, lo studente deve scrivere il codice per descrivere un filtro efficace per gli attacchi e usarlo come torre di difesa della CPU.

Il tool segue la meccanica dei *tower defense game* per imparare a scrivere le specifiche formali delle varie funzioni Java, mascherando l'attività con un'impalcatura molto allettante: costruire un filtro per proteggere la CPU sotto attacco.

Lo studente deve ricostruire la formula corretta assemblando dei blocchetti (ognuno dei quali può rappresentare una variabile, una costante o un operatore) utilizzando i fili come elemento di connessione, in modo da ricostruire correttamente la formula.

Il sistema riconosce se la specifica sia equivalente a quella fornita dal professore, facendo una verifica semantica della formula scritta ed eventualmente, a discrezione dell'insegnante, anche un test random: si assegnano gli stessi valori casuali alle espressioni del professore e dello studente controllandone nuovamente l'equivalenza.

Questo tool è stato testato solamente in un modo preliminare su un corso di *Software Testing and Verification*, dando comunque buoni risultati per quanto riguarda il coinvolgimento degli studenti. Gli aspetti che non hanno convinto i soggetti del test sono legati maggiormente alla comprensione delle meccaniche di gioco non abbastanza chiare da principio (può essere dovuto al tutorial troppo breve o al design troppo complesso).

Va citata infine una delle prime applicazioni della Gamification allo sviluppo del software che riguarda tutte le fasi dello sviluppo: **HALO** (Highly Addictive, sociaLly Optimized) analizzato nello studio di (Sheth et al., 2011).

HALO è uno strumento utilizzato per inserire una componente caratteristica dei giochi *MMORPG* (Massively Multipayer Online Role Play Game) agli IDE più comuni tramite l'installazione di un apposito plugin. Questo plugin utilizza un elemento peculiare: le *quest*, atte a mascherare i task di tutti i giorni dietro ad un "sipario ludico", utilizzando anche un sistema di ricompense per i compiti portati a termine che comprende punti esperienza, valuta virtuale, titoli e vere ricompense.

Le quest comprendono task che hanno uno spettro di difficoltà molto variabile: esistono quest che comprendono un solo compito introduttivo di complessità minima, quest che comprendono molti incarichi completabili da una sola persona e quest che coinvolgono necessariamente più persone per il proprio compimento. Inoltre, le quest possono essere divise in sub-quest che possono essere completate indipendentemente oppure legate tra loro da un particolare ordine. Le sub-quest possono inoltre essere obbligatorie per il completamento della quest principale o opzionali, nelle quali è lasciata al giocatore la libertà di portarle a compimento per ottenere un maggior punteggio o una ricompensa extra.

Gli elementi ludici presenti hanno diverse funzioni:

- i *punteggi* servono semplicemente ad indicare l'esperienza del giocatore e il suo livello;

- la *valuta virtuale* può essere spesa per fare acquisti all'interno della piattaforma per personalizzare il proprio alter ego;

- i *titoli* sono soprannomi esclusivi che si possono attribuire al proprio alter ego solo dopo aver conseguito un traguardo peculiare (ad esempio "The Bug Slayer" si potrebbe sbloccare dopo aver fatto report e fix di un numero elevato di bug);

- le *vere ricompense* sono degli omaggi che si possono ottenere in cambio di un numero elevato di monete virtuali o al conseguimento di particolari eventi (alcuni esempi possono essere una gift card in cambio di un grosso numero di monete virtuali o ancora un pranzo offerto dall'azienda per tutto il team che ha raggiunto un traguardo importante come il porting dell'applicazione da un sistema operativo ad un altro).

1.2.4 Applicazioni nel Software Testing

Come già detto in precedenza, una delle fasi più dispendiose dello sviluppo di un software in termini di tempo e di risorse è quella di testing. Nonostante sia cruciale avere una test suite completa ed esaustiva per poter riconoscere e prevenire le problematiche derivanti da bug sparsi nel codice, spesso il software viene testato in maniera blanda e tutt'altro che esaustiva per poter rispettare le scadenze imposte, trascorrendo maggior tempo nella fase di scrittura del codice. In aggiunta la fase di testing è considerata spesso noiosa e ripetitiva e non in grado di stimolare l'attenzione del tester.

Per rendere più piacevole tale disciplina sono stati fatti vari tentativi, tra cui l'applicazione della Gamification. Di seguito vengono presentati alcuni esempi.

Un primo esempio di tool che applica la Gamification ad un'attività di testing è **Code Defenders** ideato da (Rojas & Fraser, 2016). Questo strumento ludicizza l'attività di mutation testing rendendola interattiva e divertente. Il linguaggio predefinito è Java, ma si può estendere ad altri linguaggi.

Il sistema definisce due ruoli: gli *attaccanti* che devono introdurre dei bug nel codice per mettere in luce fragilità della test suite esistente e i *difensori* che devono invece migliorare la test suite arricchendola di test supplementari al fine di renderla più robusta. L'attaccante, apportando delle modifiche al codice, crea così un *mutante*; se un mutante riesce a superare tutti i test della test suite originale, ma fallisce dei test nella nuova test suite arricchita dal difensore, allora il difensore ottiene un punteggio. L'attaccante, viceversa, fa punti se il mutante creato non è stato rintracciato con nessun fail sulla test suite arricchita. Il punteggio assegnato dipende dal numero di test che un mutante passa.

Alcuni mutanti sono sintatticamente differenti ma semanticamente uguali al codice da testare originale: questi sono detti mutanti *equivalenti* e non sono considerati validi nel sistema ludicizzato. Se il difensore sospetta che l'attaccante abbia prodotto un mutante equivalente invece che uno normale può invocare il *duello*. Se l'attaccante è sfidato a duello deve dimostrare a sua volta di aver prodotto un mutante non equivalente creando una test suite nuova con dei test che facciano fallire tale mutante; se l'attaccante non è in grado di produrre tale test suite, allora il sistema considera il mutante equivalente e tutti i punti raccolti dall'attaccante sono persi.

Code Defenders ha un'implementazione web-based e può essere giocato sia in coppia che in gruppo. È anche possibile giocare in modalità singola, ma necessita l'integrazione di un tool di generazione dei test e un tool di generazione automatica dei bug.

Code Defenders è stato impiegato come strumento per insegnare software testing a diversi livelli accademici con esiti molto incoraggianti (Fraser et al., 2019). Inoltre, è emerso come la test suite risultante sia molto più robusta rispetto a quelle generate automaticamente coi tool esistenti.

L'introduzione di questa piattaforma in un corso universitario deve essere graduale e permettere agli studenti di familiarizzare con l'ambiente prima di approfondire gli argomenti. Le classi Java su cui creare test suite e mutanti devono essere scelte dal professore in modo che abbiano una complessità crescente ma sempre bilanciata: sbilanciare la test suite o la classe in esame può favorire una delle due parti che si sfidano.

Le classi Java usate nello studio sono state prima una introduttiva creata da zero, poi due classi matematiche con metodi noti agli studenti, successivamente si è passato ad altre classi esistenti nel mondo open source. Particolarmente interessanti

sono le classi che rappresentano strutture dati in quanto dotate di poche dipendenze di codice o completamente prive.

Le coppie sono state formate in modo da essere le più bilanciate possibile: gli studenti che avevano ottenuto un punteggio simile nella stessa sessione laboratorio venivano accoppiati nel laboratorio successivo. Sono state anche svolte sessioni in cui non ci si esercitava a coppie, ma in gruppi di attacco contro gruppi di difesa, alcune delle quali sono state eseguite con numeri non bilanciati, basandosi sulla "forza" degli studenti (studenti più deboli raggruppati a difendere un attaccante molto forte).

Una successiva applicazione della Gamification ad un altro aspetto del software testing si può trovare nello studio di (Santos et al., 2019) che presenta **Clean Game**: un tool per ludicizzare la pratica di *code refactoring*.

Il *code refactoring* è una sotto-attività del software testing che ha lo scopo di individuare le fragilità del codice e sostituirle con un codice equivalente dal punto di vista funzionale, ma privo di tali imprecisioni: queste fragilità vengono chiamate in gergo *code smell*. I *code smell* sono artefatti noti assieme al proprio metodo di refactor, definiti in letteratura (Fowler, 1999); in Clean Game si sfrutta un analizzatore statico di codice sorgente chiamato **PMD**, che effettua il parsing del codice dato in input alla ricerca dei *code smell* noti.

Clean Game è diviso in due moduli: un quiz sui *code smell* a risposta multipla e un modulo di identificazione dei *code smell*. Gli elementi di Gamification usati in questi moduli sono: player status, punteggi, badge, il tempo impiegato (per fornire le risposte nei quiz e nella ricerca dei code smell) e le classifiche per ogni modulo aggiornate in tempo reale. Sono utilizzabili fino a tre suggerimenti per l'identificazione dei code smell e per i quiz, tuttavia chiedere aiuto alla piattaforma penalizza il giocatore in termini di punteggio.

Clean Game può essere adottato non solo a livello accademico, ma anche a livello aziendale in quanto i quiz vengono creati dinamicamente in modo automatico sulla base del codice fornito in input e, non essendoci la necessità di una figura intermedia che conosca il codice e i suoi code smell, l'integrazione con l'ambiente lavorativo può essere realizzata con costi contenuti.

Clean Game può inoltre essere completamente integrato con la GitHub API: alla creazione di una nuova stanza l'utente deve fornire l'URL della repository Java su GitHub su cui si vuole operare. Dopodiché il codice è clonato e trasformato

in un Abstract Sintax Tree (AST) usato come oracolo per le domande relative al codice.

L'esperimento condotto da (Santos et al., 2019) mostra come l'uso di tale sistema ludicizzato aiuti nell'identificazione dei code smell e nel refactor, portando ad una individuazione del doppio dei code smell rispetto alla loro ricerca in un ambiente non ludicizzato. Dalle valutazioni personali dei partecipanti sono emersi dati positivi di coinvolgimento eccetto che negli aspetti di focus (poca concentrazione potrebbe essere tuttavia dovuta all'eccessiva difficoltà dell'esperimento) e nella soddisfazione: i risultati sono stati migliori, ma è probabile che la Gamification non abbia avuto un design abbastanza coinvolgente da migliorare significativamente l'entusiasmo e il coinvolgimento dei partecipanti.

Va tuttavia sottolineato come il campione cui è stato somministrato l'esperimento fosse molto ristretto (18 persone) e limitato all'ambiente accademico; non è dunque noto ad oggi se a livello aziendale l'impatto possa essere lo stesso.

In un ulteriore studio (Costa & Oliveira, 2019), invece, si descrive un possibile tentativo di applicazione dei princìpi della Gamification all'insegnamento dell'exploratory testing, ossia un approccio di testing manuale che pone enfasi sulla libertà e la responsabilità del tester di esplorare il sistema.

L'attività di testing viene immersa in un'ambientazione piratesca il cui tema principale è quello della caccia al tesoro: il codice viene diviso in aree da esplorare e i tester devono adottare delle strategie per scovare i tesori (ossia i bug).

Vengono utilizzati diversi elementi:

- *Profili*: identificano i diversi ruoli in gioco, come tester o esperti (ossia coloro che gestiscono il sistema di Gamification);

- *Avatar*: rappresenta lo studente nel gioco e può raggiungere diversi livelli a seconda del punteggio ottenuto; è inoltre presente un *general avatar* che rappresenta un team di studenti, ottenuto sostanzialmente facendo la media fra gli avatar dei componenti.

- *Attività*: sono i vari task che un tester può svolgere. Al completamento di ognuna di esse si ottiene un determinato *punteggio*. Ottenere un certo numero di punti durante le fasi del gioco può portare il tester ad ottenere delle

medaglie (ossia dei badge) o delle *monete* (chiamate *bitskull*), che costituiscono il premio in caso di raggiungimento del punteggio massimo in una fase e permettono di comprare delle risorse (come le *carte*);

- *Carte*: sono oggetti equipaggiabili all'avatar di un tester. Sono disponibili tre tipi di carte:
  - *Difesa*: forniscono uno strumento per difendersi da eventuali attacchi nemici;
  - *Attacco*: permettono di attaccare i nemici (per esempio consultando i loro requisiti di sistema, parte del tutorial, ricevendo l'aiuto di uno specialista ecc.);
  - *Accessori*: permettono di personalizzare il proprio avatar.

All'inizio del corso, gli studenti (divisi in team) sono stati istruiti sulle dinamiche del sistema ed è stata loro raccontata una breve storia sull'ambientazione in modo da rendere il tutto più immersivo. Le attività svolte successivamente sono state invece suddivise nelle seguenti fasi (spiegate descrivendo uno scenario esemplificativo con due soli team):

1. Al team A e al team B è stato lasciato un certo lasso di tempo per cercare il maggior numero di bug possibile nel proprio codice;

2. Entrambi i team hanno prodotto un report (*bug report*) su quanto fatto fino a quel momento, descrivendo le strategie adottate ed eseguendo una prioritizzazione dei bug trovati;

3. Si è svolta una fase di *battaglia*, dove i due team si sono scambiati i report prodotti nella fase 2, giudicandoli a vicenda e inserendo i risultati dell'analisi in un altro report (*analysis report*);

4. Un esperto ha infine giudicato i report prodotti dai due team, assegnando un punteggio in base al numero e alla qualità dei bug segnalati nel bug report e producendo un'analisi di tale resoconto, così da confrontarla con l'analysis report prodotto dall'altro team e assegnargli dunque un determinato punteggio.

Durante la consegna dei risultati, i team, in base al loro punteggio, potevano ricevere una ricompensa ed eventualmente una carta (previa risoluzione di un puzzle aggiuntivo). I team il cui general avatar è stato portato all'ultimo livello, dopo la risoluzione di tre puzzle, potevano essere ricompensati con un premio nascosto fisicamente all'interno della classe. Ciò ha enfatizzato ancora di più l'atmosfera e il tema della caccia al tesoro, aggiungendo immersività.

Il risultato di questo approccio può essere valutato sulla base dei dati ricavati dalle varie attività, del feedback degli studenti e del confronto fra due form che devono essere sottoposti agli studenti prima e dopo il corso, per permettere di capire come siano cambiate le loro conoscenze grazie al corso stesso e all'utilizzo di un sistema gamificato.

1.2.5 Limiti

I casi di applicazione delle tecniche di Gamification hanno avuto una crescita significativa negli ultimi anni (Pedreira et al., 2015), tale da stimolarne un ampio utilizzo in ambito IT. In molti casi, però la realizzazione e l'integrazione dei sistemi ludicizzati sono state eseguite in maniera inadeguata, evidenziando i limiti di applicabilità di questo approccio. Di seguito verranno elencati alcuni casi noti.

Nello studio realizzato da (Berkling & Thomas, 2013) citato in precedenza, si evidenzia come dai feedback raccolti direttamente dagli studenti tramite dei questionari, sia emerso una profonda disparità nelle aspettative riguardanti l'uso dei termini e degli elementi della Gamification in un ambito come quello universitario. Molti studenti (la maggior parte dei quali non definiva sé stessa "gamer") individuò infatti nel semplice divertimento la principale ragione che li spinge a giocare solitamente, trascurando quasi completamente altri fattori, come ad esempio il senso di sfida, la sensazione di autonomia o la voglia di raggiungere degli obiettivi, che sono invece tra gli aspetti principali su cui fa leva la Gamification. Il risultato è stato quindi che la maggior parte degli studenti ha percepito tutti gli elementi ludici come inutili, noiosi o addirittura dannosi in certi casi.

Oltre a questo aspetto, sono state riscontrate anche delle problematiche legate all'eccessivo senso di autonomia generato negli studenti, che ha portato diversi di loro a pensare di non aver più bisogno dell'ausilio dei professori o addirittura di non aver necessità di seguire il corso stesso. Molti di loro, inoltre, avevano trascurarono

del tutto la maggior parte degli elementi maggiormente legati al gioco, dai più classici, come classifiche, punti, badge e livelli, ai meno comuni, come i punti extra e il riconoscimento pubblico assegnati in caso di aiuto ad altri studenti.

Gli sviluppatori della piattaforma hanno individuato, tra le principali cause dietro al fallimento dell'esperimento, la mancanza di appeal della piattaforma stessa dal punto di vista estetico, giungendo inoltre alla conclusione che la scelta migliore fosse quella di integrare gli elementi della Gamification nel sistema senza nominarli esplicitamente e di introdurre i cambiamenti in maniera più lenta e graduale.

Secondo lo studio di (Dal Sasso et al., 2017) uno dei grandi problemi dei sistemi ludicizzati è la *pointsification* ovvero la ricerca da parte dell'utilizzatore del sistema ludicizzato di totalizzare un grande quantitativo di punti, trascurando il reale scopo dell'introduzione dei punteggi: il completamento di un determinato lavoro.

Il giocatore finisce così a trascurare i compiti più difficili e più remunerativi in termini di punteggio in favore di quelli più semplici. Questo comportamento è dovuto al fatto che i giocatori focalizzati sull'attività di raccoglimento di punti, pur di avere una assegnazione di punteggio immediata, concentrano la loro attenzione sulle attività meno dispendiose a livello di impegno, risorse e tempo per non dover aspettare il completamento di un task molto complesso.

Un ulteriore aspetto spesso trascurato quando si tenta di fare una analisi costi/-benefici sull'uso della Gamification è il fatto che non esiste nessuna ricerca sui suoi effetti a lungo termine. Si ipotizza (Platonova & Bērziša, 2017) che a lungo andare si perda l'impatto positivo mostrato sul breve periodo, ma tale affermazione resta un'ipotesi che non può essere né confermata né smentita allo stato attuale dell'arte.

Altre volte il sistema non porta vantaggi perché semplicemente male implementato: è fondamentale che il layer di Gamification, oltre che essere appetibile dal punto di vista ludico e dell'intrattenimento, sia anche completamente privo di bug e glitch, per evitare che tali problemi compromettano l'esperienza dell'utente, rendendola meno gratificante e immersiva.

Nell'esperimento condotto da (Buckley & Clarke, 2018) si fa notare come una delle possibili ragioni del fallimento del sistema didattico ludicizzato proposto sia l'insorgenza di bug notata proprio dagli studenti appartenenti al gruppo sperimentale cui era sottoposta la piattaforma ludicizzata. Oltre a questa ragione viene menzionato anche il fatto che il gruppo di controllo potesse accedere al materiale fornito dal professore senza dover passare attraverso un meccanismo di identificazione, mentre al gruppo sperimentale era richiesta un'identificazione iniziale tramite

login con username e password.

Altri sistemi ludicizzati hanno aspetti di applicabilità limitata: funzionano molto bene a livello accademico per l'insegnamento di una particolare attività, ma trovano difficoltà nell'applicazione in un ambiente aziendale. Tali difficoltà possono essere dovute alla complessità di integrazione con l'ambiente di lavoro (se un tool non è ben integrato porta alla distrazione dell'utilizzatore, che è forzato a fare un context switch costoso da un'attività ad un'altra perdendo concentrazione) (Prasetya et al., 2019) oppure al fatto che i tool, per essere applicati, necessitano di un overhead che, se in ambito accademico è demandato al professore, nell'ambiente lavorativo non ha un'attribuzione chiara e univoca: l'assegnazione del punteggio fatta automaticamente deve tenere conto della difficoltà del task (non banalmente quantificabile) e il completamento delle quest non sempre può essere determinato in modo automatico, ma necessita che sia un persona fisica a considerare una quest completa (Sheth et al., 2011).

1.3 GUI Testing e Tool Esistenti

Il GUI testing è la disciplina che si occupa di testare l'applicazione utilizzando l'interfaccia grafica, interagendo proprio come farebbe l'utente finale. Esistono principalmente due approcci al GUI testing: manuale e automatizzato.

Il metodo **manuale** richiede che sia il tester a effettuare i controlli eseguendo i test case a mano ad ogni iterazione; ciò comporta un costo molto elevato soprattutto in termini di tempo speso nell'esecuzione dei test, unito ad una maggiore probabilità di commettere errori durante la sessione, essendo l'esecuzione fortemente legata a fattori umani.

L'approccio **automatizzato**, invece, si basa sulla creazione di script di test da parte del tester e sulla loro esecuzione automatica ad ogni iterazione. Gli script possono essere programmati direttamente in un linguaggio (o pseudolinguaggio) di programmazione oppure creati in modalità *Record & Replay*, in cui il tester registra le operazioni definite nel test case una tantum, eseguendole manualmente. Successivamente, ad ogni iterazione, le interazioni registrate vengono emulate.

Questa modalità permette di risparmiare in termini di tempo e di risorse umane impiegate (Borjesson & Feldt, 2012), ma anche di ottenere prestazioni migliori in termini di coverage raggiunta e bug trovati. Il rovescio della medaglia è dato da una minor flessibilità e una forte fragilità dei test case automatici: infatti ogni volta

che la GUI subisce alterazioni, talvolta anche minime, i test potrebbero necessitare l'intervento umano per riprogrammarli. Inoltre, sebbene l'esecuzione automatizzata dei test sia più rapida se confrontata con l'esecuzione manuale, la creazione e la manutenzione dei test richiede molto tempo.

Una nuova tipologia di testing automatizzato è l'**Augmented Testing** (AT), introdotto nello studio di (Nass et al., 2019) basandosi sulla tecnica già nota di *Record & Replay*. Questo nuovo metodo permette di svolgere l'attività di creazione della test suite del *System Under Test* (SUT) usando la cosiddetta Augmented GUI: un'interfaccia che si sovrappone alla GUI del sistema, arricchendola con informazioni supplementari dedotte a partire dai metadati. La Augmented GUI fornisce suggerimenti al tester, evidenzia aree importanti dell'interfaccia e permette di registrare dei commenti.

Il principio su cui si basa questa tecnica è l'utilizzo di un *Augmented Layer* (AL) che si interpone tra il tester e il sistema testato e mediando le interazioni che avvengono fra queste due entità. L'AL riceve le informazioni sulla pagina da mostrare dal SUT ed elabora i metadati presenti, fondendo in una sola vista l'interfaccia originale con le informazioni aggiuntive dedotte (bottoni da cliccare, testo da controllare, caselle di testo da riempire ecc.).

Il tester interagisce con l'AL attraverso check, difetti e azioni aumentate che captano l'interazione avvenuta e agiscono sul modello riproducendo a tutti gli effetti la stessa azione sull'interfaccia originale. Tutte le interazioni avvenute sono registrate durante la sessione ed evidenziate da un rettangolo colorato attorno al widget su cui è avvenuta l'interazione e il tester può selezionare un'azione precedentemente registrata per riprodurla oppure crearne una nuova.

L'Augmented Testing precedentemente descritto è implementato da **Scout**: un prototipo sviluppato in Java che applica tali nozioni ad una applicazione web. Tale prototipo ha alcune limitazioni nella presente implementazioni che verranno esposte nei capitoli successivi.

Negli studi di (Nass et al., 2019) e (Nass, Alégroth, & Feldt, 2020) viene mostrato come l'Augmented Testing abbia delle potenzialità elevate in quanto permette di creare test case in meno tempo e con più semplicità rispetto a tool ad oggi esistenti. Combinando gli effetti della Gamification a questa tecnica si mira a creare un ambiente di GUI testing efficiente, facile da imparare e piacevole per l'utilizzatore, al fine di ottenere un miglioramento delle prestazioni e della soddisfazione del tester.

Nelle sezioni successive si parlerà dello sviluppo di un plugin per Scout che implementi tecniche di Gamification e verranno valutati gli effetti sortiti.

1.4 Obiettivi

L'obiettivo principale del presente lavoro di tesi è quello di fornire un tentativo di applicazione dei concetti di Gamification ad un ambito nuovo, quello del GUI testing. I risultati che si cercherà di ottenere comprendono il miglioramento della percezione del tester della disciplina, creando un ambiente capace di trasmettere soddisfazione e interesse nel lavoro svolto, unito ad un potenziamento delle performance riscontrate nelle sessioni di testing, ottimizzando il tempo e i costi che tale disciplina necessita.

Il resto della tesi è strutturato come segue:

- Il Capitolo 2 fornisce una descrizione dettagliata riguardo all'implementazione degli elementi di Gamification inseriti nel tool;
- Il Capitolo 3 fornisce un esempio di sessione di testing svolta utilizzando Scout;
- Il Capitolo 4 descrive la sperimentazione e il processo di validazione del plugin;
- Il Capitolo 5 analizza le limitazioni riscontrate e riassume infine gli spunti aperti per lavori futuri.

Capitolo 2

Architettura e design

Per costruire un plugin che implementi la Gamification è stato necessario analizzare il software già esistente nel dettaglio.

2.1 Scout

Scout (Nass et al., 2020) è un prototipo di tool di GUI testing, che implementa il concetto di Augmented Testing, sviluppato per valutare l'efficacia della tecnica in collaborazione con professionisti del settore industriale. Gli sviluppatori mantengono il core del tool confidenziale, dando però la possibilità di arricchire con nuove funzionalità il software tramite lo sviluppo di plugin. Alcuni di essi, nonostante siano di fatto opzionali, sono alla base del corretto funzionamento del software.

In Figura 2.1, viene mostrata una panoramica del tool, comprensiva dei moduli già presenti nel codice originale (non colorati all'interno del grafico), di quelli già presenti ma modificati o adattati per permettere l'implementazione degli elementi ludicizzati (evidenziati in arancione) e, infine, dei moduli aggiunti ex novo (evidenziati in verde).

Come si evince, Scout memorizza al proprio interno un *Application State* che modella lo stato del SUT con tutte le dipendenze interne necessarie al funzionamento e le varie interazioni avvenute. A partire dall'home state viene creata una struttura che rappresenta lo state model Figura 2.2, contenente gli stati visitati, che può essere attraversato e modificato man mano nelle operazioni di testing.

L'*Augmented GUI* (AGUI) è l'interfaccia tramite la quale il tester agisce sull'app da testare. Mentre l'interfaccia col web di Scout nasconde la GUI del SUT,

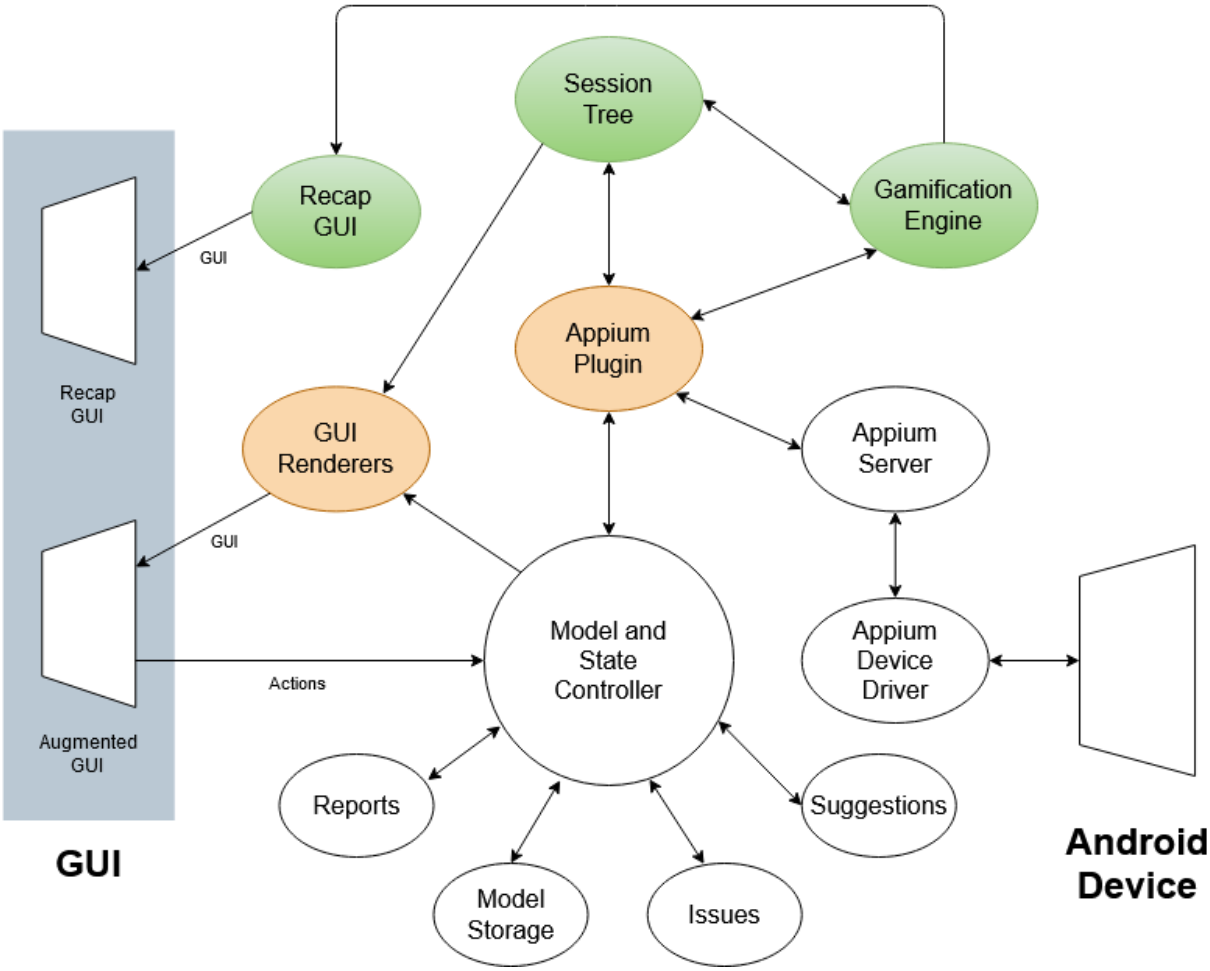

Figura 2.1. Panoramica del tool.

l'interfaccia con Android è basata proprio sull'aspetto originale dell'app mostrata nell'emulatore: rimane infatti in background la finestra dell'Android Virtual Device e periodicamente si acquisisce uno screenshot dell'emulatore, necessario per replicarne il contenuto nella finestra di Scout. Il sistema di interfacciamento con Android è realizzato mediante un plugin che prevede l'inizializzazione di un Server Appium per realizzare l'accesso al dispositivo emulato. Il driver, infatti, non mostra direttamente il risultato dell'emulazione nella finestra principale di Scout, è invece necessario che la schermata, catturata tramite Appium, venga salvata in un file locale e poi riportata nell'AGUI. Questi passaggi, che non avvengono quando il SUT è web based grazie alla mediazione diretta del webdriver, rendono più complesso il meccanismo di funzionamento di Scout per Android; tale complessità si concretizza

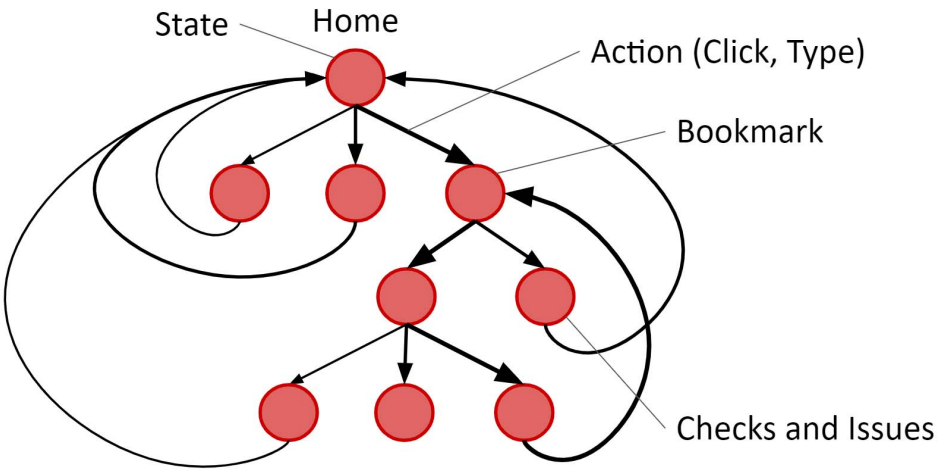

Figura 2.2. State Model.

con un ritardo piuttosto elevato nell'esecuzione dei comandi e intercettazione del feedback visivo, costituendo un problema non banale.

Contemporaneamente alla visualizzazione dello screenshot avviene un parsing del file XML della pagina sorgente, tramite il quale la logica deduce la presenza e la disposizione dei widget presenti nella pagina. Sono proprio i widget riconosciuti in questo modo che permettono all'AL di captare l'interazione del tester e di riprodurla fedelmente nell'interfaccia originale.

Le informazioni presenti quindi nell'AGUI sono quindi provenienti da due sorgenti: quella visiva che permette di ricostruire l'interfaccia di partenza e quella descrittiva che permette di dedurre le informazioni necessarie per le interazioni.

L'AGUI arricchisce la GUI del SUT con elementi aggiuntivi come suggerimenti e azioni aumentate (check, issue, click, ecc.), che vengono renderizzati tramite la classe Java *AugmentState* e *AugmentToolbar*.

La prima riceve le informazioni sullo stato del SUT dallo State Model e traduce i metadati in oggetti visivi che contornano le zone della GUI in cui sono avvenute le interazioni con un colore dipendente dal tipo dell'interazione stessa: i check sono contornati in verde, i click in blu, i difetti in rosso e le espressioni che devono ancora essere valutate in giallo.

La seconda permette al tester di configurare Scout, scegliendo i plugin da attivare, e di gestire la sessione corrente di testing, avviandola o interrompendola. Nella versione di Scout per Android non è possibile salvare un particolare stato per

un caricamento successivo, né è possibile navigare all'indietro il grafo degli stati poiché, a differenza di quello che accade con un sito web, dove ogni URL può essere caricato in qualsiasi momento, in una applicazione Android il passaggio da una pagina mostrata a schermo ad un'altra non può avvenire deliberatamente, ma deve seguire delle regole ben note che variano da app ad app (tap su un bottone, swipe sulla pagina, eventi esterni ecc.).

Tutte le azioni catturate dalla AGUI sono salvate nello State Model, implementato come grafo pesato, in cui ogni nodo rappresenta un particolare stato dell'applicazione. I nodi sono tra loro collegati tramite le azioni eseguite dal tester che, modificando lo stato, determinano un avanzamento nel grafo. I vari stati sono memorizzati all'interno del componente State Controller, che tiene traccia dell'intera struttura. Mantenere la sincronizzazione tra lo stato del SUT e il nodo corrente del grafo è tutt'altro che banale, poiché in una app mobile non è noto a priori quali siano tutte le azioni che modificano lo stato corrente; nondimeno la sincronia è particolarmente importante, poiché tutte le informazioni riguardanti l'applicazione, che sono essenziali sia per il funzionamento del plugin di Gamification che di Scout stesso, sono memorizzati proprio in State Controller.

2.2 Appium

L'elemento essenziale che permette la congiunzione tra l'app Android in esecuzione nell'ambiente emulato e Scout è l'**Appium** (Verma, 2017).

Appium è un framework per il software testing di applicazioni mobile open-source e multipiattaforma. Questo strumento è particolarmente utile perché permette l'esecuzione di test suite in modo indipendente dalla piattaforma di esecuzione oltre che a testare applicazioni create in maniera differente (applicazioni native, ibride e web app); può per tali motivi essere usato per testare un ampio spettro di applicazioni Android e iOS senza cambiare il core del codice.

Appium è basato su una architettura Client-Server in cui i vari componenti sono rappresentati in Figura 2.3.

Il componente principale è **Appium Server**, un software scritto in Node.js che permette di interpretare ed elaborare le richieste ricevute dall'Appium Client e di trasformarle nelle corrispondenti invocazioni dell'Appium Device Driver, che esegue le istruzioni a basso livello con l'ambiente desiderato (Android/iOS su dispositivi

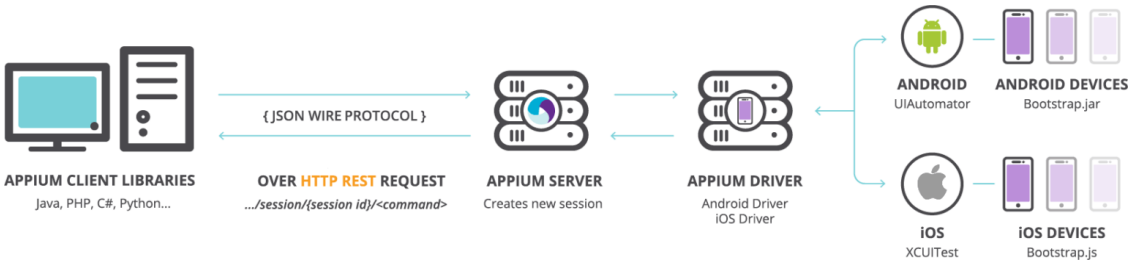

Figura 2.3. Architettura di Appium

fisici o simulati). L'Appium Server è in grado di creare diverse sessioni su dispositivi diversi in contemporanea.

Le **Librerie del Client Appium** sono disponibili per diversi linguaggi di programmazione, tra cui il caso di interesse, Java, e permettono di segnalare all'Appium Server di creare, arrestare la sessione e ricevere i risultati intermedi. Per richiedere la creazione di una sessione è necessario preparare un'opzione di *Desired Capabilities* in cui vengono specificati i dettagli dell'ambiente di esecuzione peculiari della sessione da creare (la piattaforma di esecuzione, il nome del dispositivo, la locazione, il nome del SUT, ecc.). Grazie a tali informazioni, l'Appium Server è in grado di predisporre la sessione desiderata, scegliere la giusta applicazione da testare, utilizzando il corretto Device Driver.

La comunicazioni tra i due componenti avviene tramite un **protocollo JSON Wire**, un protocollo standard che è agnostico rispetto al linguaggio di programmazione della libreria del Client: in particolare, si utilizza una HTTP REST request con un formato JSON in input. I dati degli oggetti che devono essere trasmessi al Server sono convertiti in formato JSON e viceversa utilizzando metodi di serializzazione e deserializzazione. Essendo uno strumento multipiattaforma, il Server non ha bisogno di comprendere i linguaggi di programmazione delle librerie Client, ma si limita a identificare il protocollo per creare e gestire le sessioni.

Infine, la componente di più basso livello di Appium sono gli **Appium Device Driver** o semplicemente Appium Driver, la parte del software che gestisce realmente la connessione a basso livello con l'ambiente mobile con cui Appium Server deve interagire. Basandosi sull'istanza di *Desired Capability* ricevuta, viene contattato dal Server il driver corretto per la connessione all'app da testare. Nel caso dell'emulazione di Android, è *UiAutomator2* a contattare *bootstrap.jar*: un software in

esecuzione che si comporta come un Server TCP, ricevendo e inoltrando al dispositivo Android i comandi necessari all'esecuzione dei test e inviando ad Appium Server le informazioni sullo stato della sessione in corso.

2.3 Descrizione dei file di Log

Come punto di partenza dello sviluppo del plugin è stato creato un sistema di logging che annotasse ogni interazione, descrivendola nel contesto della pagina in cui si verifica, arricchendola inoltre con alcuni metadati temporali e altri legati ai widget. Il modello descritto nella sezione 2.4 è stato sviluppato in seguito all'analisi dei file di log prodotti nelle sessioni di esempio, svolte utilizzando il tool "as is", in particolare esaminando le interazioni raccolte e producendo manualmente le quantità grezze in seguito descritte.

Vengono di seguito descritti i dati prodotti dal logger alla fine di ogni sessione di GUI testing.

Per ogni pagina viene mostrato lo stato che la identifica, il tempo totale in millisecondi che il tester ha trascorso in tale pagina e infine l'elenco delle interazioni avvenute, rappresentate tramite delle stringhe formattate nel seguente modo:

<tipoInterazione> <idInterazione> <timestamp>

tipoInterazione specifica la tipologia dell'interazione registrata, che può essere una tra le seguenti:

- *Check*: Indica che il tester ha richiesto un controllo sul contenuto del widget.
- *Issue*: Indica che il tester si aspettava qualcosa di diverso dal widget selezionato.
- *Click*: Registra un click su un widget. Solitamente questo porta al caricamento di un'altra pagina web.
- *Type*: Annota la scrittura di una stringa di testo all'interno di una casella di testo. Oltre alla tipologia di interazione, il logger annota anche la stringa digitata dall'utente.
- *Select*: Annota la selezione di una tra le opzioni nella drop-down list; ogni volta che si cambia l'elemento selezionato viene prodotta una interazione di questo tipo.

Dopo aver specificato la tipologia dell'interazione registrata viene indicato **idInterazione**, ossia un identificativo del widget Android sul quale è avvenuta l'interazione. L'assegnazione di una stringa univoca all'interno della pagina si basa sui metadati assegnati all'elemento nel file XML.

Le tipologie di elementi che compongono una pagina in Android sono innumerevoli, tra questi distinguiamo quelli standard, quelli definiti in particolari librerie e infine quelli definiti ex novo dal programmatore. Allo stesso modo, anche la presenza e la descrittività dei campi che compongono il tag XML sono peculiari per ogni app e dipendono dalla funzione dell'elemento all'interno dell'app e dall'uso che ne viene fatto dallo sviluppatore.

Questa variabilità impone la necessità di stabilire una priorità nell'assegnazione dell'identificativo, seguendo questo ordine di attribuzione:

1. Campo ID (se disponibile).
2. Nome del widget (se assegnato).
3. Codice hash della stringa ottenuta concatenando il campo value e il campo text (se almeno una delle due è disponibile).
4. Classe Java del widget (se nessuna delle condizioni precedenti è soddisfatta).

Infine, l'ultima componente del log di un'interazione è **timestamp**: l'istante di tempo alla quale essa avviene, espresso in millisecondi.

2.4 Rappresentazione della Sessione

All'avvio del software è possibile iniziare una sessione di testing con l'apposito pulsante, indicando un ID che rappresenti il tester che vuole effettuare la sessione.

I parametri necessari ad Appium per la connessione con l'emulatore di Android devono invece essere scritti in un file chiamato *AppiumConf* posizionato nella cartella *Gamification*, all'interno della home directory di Scout. Tali parametri devono essere scritti uno per riga nel file e seguire il seguente ordine:

1. Nome dell'AVD : il modello dell'AVD (Android Virtual Device) da utilizzare.
2. Nome dell'Emulatore: il nome del device emulato da utilizzare; si può visualizzare tramite il comando

```
adb devices
```

3. Nome della piattaforma: Android in questo particolare caso, ma anche iOS sarebbe valido in futuri sviluppi del tool.
4. Versione della piattaforma.
5. Nome del package: il package name in cui va localizzata l'applicazione Android.
6. Nome della main Activity: il nome dell'Activity Android che deve essere usata per lanciare l'applicazione.

Si noti che, nonostante l'ID del tester sia di default reso facoltativo da Scout, il plugin di Gamification salva le informazioni relative alla sessione appena conclusa soltanto se tale ID è stato correttamente inserito.

La rappresentazione di una sessione di testing è resa possibile dall'introduzione di quattro classi fortemente interconnesse, raggruppate nel package rappresentato in Figura 2.4. La singola sessione è stata rappresentata tramite un oggetto di classe *Session*, che modella una struttura dati ad albero non binario. Ogni nodo dell'albero è un oggetto di classe *Node*, ossia un wrapper di una pagina, unita ad altri attributi che consentono la navigazione dell'albero stesso, quali il nodo padre (attributo *father*) e una lista dei nodi figli (attributo *children*); viene memorizzata inoltre la stringa che identifica l'Activity principale con il proprio package, necessaria per rilevare quando si naviga da una pagina ad un'altra nell'app. *Session* permette inoltre di gestire ad alto livello la struttura ad albero, mantenendo un riferimento *root* alla radice dell'albero ed un altro, chiamato *current*, al nodo corrente in cui si trova il tester ad un istante temporale della propria sessione di testing.

Tra i metodi presenti in *Session*, vi sono alcuni fondamentali, tra cui *newNode()* per l'aggiunta di nodi all'albero (sempre a partire dal nodo indicato dall'attributo *current*) e *getCoverage()* per visitare l'albero ricorsivamente, restituendo un array list contenente tutte le coverage raggiunte nelle pagine visitate nel corso della sessione, utile per calcolare delle metriche che verranno spiegate nelle sezioni successive. Infine il metodo *updateState()* di supporto all'aggiunta del nuovo nodo, che permette di confrontare uno stato in input con l'intero albero degli stati, determinando

lo stato risultante: se lo stato in input coincide con quello corrente significa che la pagina attiva è rimasta la stessa, se l'input era già presente in un nodo dell'albero degli stati, quest'ultimo diventa il nodo corrente e viene ripristinato lo stato di quella particolare pagina, altrimenti se l'input non era presente nell'albero, si tratta di uno stato associato ad una pagina inedita, che deve quindi essere aggiunta in un nuovo nodo nella struttura.

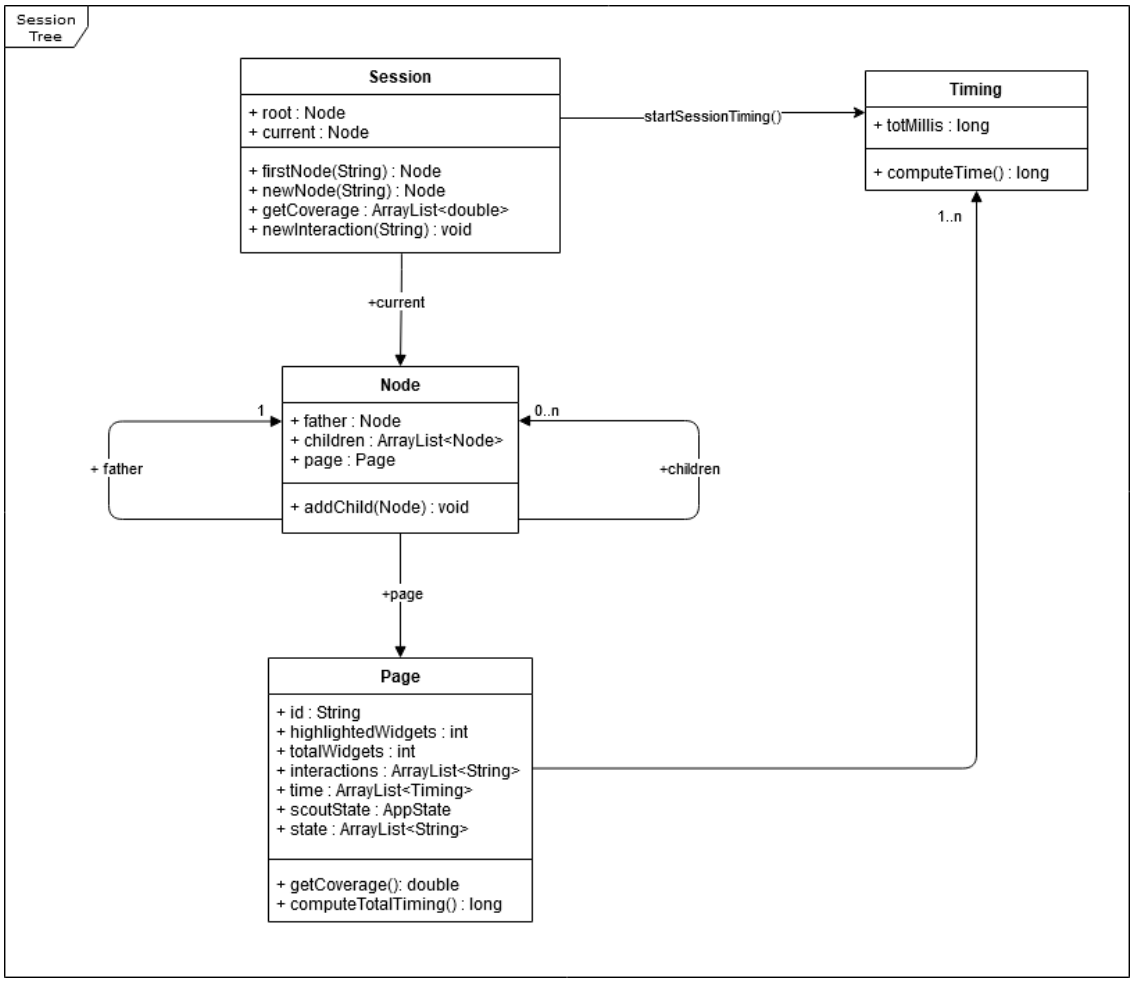

Figura 2.4. Package contenente le classi che permettono di rappresentare una sessione all'interno del plugin.

Come detto in precedenza, ogni oggetto *Node* contiene al suo interno una pagina, modellata tramite la classe *Page*. Una pagina è identificata univocamente dal proprio *stato*: la collezione dei Fragment e dell'Activity correnti. Per individuare

una pagina in modo univoco è quindi necessario associarla al Fragment Java in esecuzione, a sua volta contenuto in una Activity. La navigazione ad una pagina nuova è quindi riconosciuta dal modello proposto solamente quando cambia l'Activity in esecuzione o quando cambia l'insieme dei Fragment in essa attivi.

Per determinare lo stato attuale dell'app in esecuzione nell'AVD viene lanciato il comando:

```
adb shell dumpsys activity <nome del package>
```

Parsificando opportunamente l'output si estrae la collezione contenente l'Activity in esecuzione e i fragment attivi ad essa associati.

Si noti che lo stato di una pagina dipende da come lo sviluppatore ha implementato la stessa: in alcuni casi l'app target può essere progettata in modo da associare ad ogni schermata un'Activity differente, in altri l'Activity in esecuzione non cambia, è invece l'insieme dei Fragment ad essa associati a mutare. Se lo sviluppatore non associa esplicitamente nessun Fragment alla Main Activity, ne viene assegnato uno dal componente *ProcessLifecycleOwner* di Android: in questo modo l'insieme su cui si basa la definizione di stato proposta rimane consistente e non nullo.

Alla luce di questa scelta implementativa, esiste un caso teorico in cui, se l'app Android modifica il suo aspetto senza modificare né l'insieme dei Fragment attivi, né l'Activity in esecuzione, al mutare dell'apparenza non è associato nessun cambiamento dello stato. Questo caso può portare ad un comportamento non previsto di Scout (si veda la sezione 5.1 sui limiti di applicabilità).

Le istanze della classe *Page* contengono inoltre una stringa utilizzata per identificare la pagina, calcolata come l'hash code dello stato della pagina (attributo *id*), un riferimento allo stato di Scout nell'attributo *scoutState* (istanza della classe *AppState*) necessaria per il ripristino dello stato tramite lo State Controller quando si ritorna su una pagina già visitata, la lista delle interazioni che si sono verificate all'interno della pagina (attributo *interactions*), il numero di widget interagibili (attributo *totalWidgets*), il numero di widget coi quali il tester ha interagito (attributo *highlightedWidgets*) ed un meccanismo di annotazione del tempo trascorso all'interno della pagina, basato sull'utilizzo della classe *Timing*.

Timing modella un timer che definisce un solo intervallo temporale con un determinato inizio ed una determinata fine, senza fornire la possibilità di farlo ripartire una volta fermato. La classe mette a disposizione un metodo *computeTime()* che calcola la differenza fra l'istante finale e l'istante iniziale e ritorna il numero di

millisecondi trascorsi fra i due. Dato che una stessa pagina può essere visitata più volte nel corso di una stessa sessione, ogni oggetto *Page* contiene un array list di oggetti *Timing*, ognuno dei quali rappresenta la durata di una certa visita.

Ogni sessione viene identificata dall'ID del tester e dall'istante temporale in cui si conclude. Questo meccanismo fa sì che tutte le sessioni possano essere salvate e utilizzate come base per calcolare statistiche puntuali o aggregate. In particolare, al termine della sessione, le apposite classi *StatsComputer* (spiegata nel dettaglio nella sezione relativa al Gamification Engine) e *Timing* permettono di calcolare una serie di dati grezzi che forniscono delle informazioni riguardo alla sessione appena conclusa. Tra i dati più rilevanti troviamo:

- Numero totale di widget cliccabili individuati nelle pagine che compongono la sessione appena conclusa. Questo dato si ottiene visitando ricorsivamente l'albero che modella la sessione, sommando l'attributo *totalWidgets* di ogni pagina presente nell'albero.

- Numero totale di widget con cui il tester ha interagito durante la sessione. Anche questo dato viene calcolato visitando ricorsivamente l'albero e sommando l'attributo *highlightedWidgets* di ogni pagina visitata nella sessione.

- Numero di interazioni avvenute per ogni pagina. Tale dato si ottiene contando tutte le interazioni avvenute sui widget della pagina. A differenza del numero totale dei widget con cui si ha interagito, in questo caso si possono verificare più interazioni sullo stesso widget in momenti diversi della sessione (repliche identiche dell'azione o interazioni con parametri diversi).

- Il numero di pagine visitate, definito come la cardinalità dell'insieme delle pagine identificate dal proprio stato.

- Tempo trascorso all'interno di una singola pagina, espresso in millisecondi. Si noti che, avendo la possibilità di navigare in avanti e indietro, alcune pagine possono essere visitate più volte all'interno della stessa sessione. Per questo motivo, ogni oggetto *Page* contiene una lista di oggetti *Timing* i cui valori, sommati, restituiscono il tempo totale di permanenza all'interno di una stessa pagina nell'arco dell'intera sessione.

- Tempo totale impiegato dal tester per portare a termine la sessione. Viene calcolato inizializzando un oggetto *Timing* all'inizio della sessione (pressione

del tasto *Start*) e fermando il conteggio alla sua conclusione (pressione del tasto *Stop*). Si noti che tale valore può differire dalla somma delle tempistiche di tutte le varie pagine, poiché il timer della pagina è interrotto prima dell'emulazione del tocco sul widget e riprende al caricamento nella pagina successiva; ciò porta una discrepanza di anche diversi secondi in sessioni molto durature e con un alto tasso di esplorazione. Questa disparità si registra a seguito della complessità delle operazioni di emulazione del tocco, di caricamento della pagina e acquisizione dello screenshot.

2.5 Estrapolazione delle metriche

Utilizzando e combinando i dati grezzi precedentemente riportati, si può estrapolare una serie di metriche che descrivono la sessione. Tali metriche sono alla base dell'utilizzo degli elementi di Gamification introdotti nel plugin.

Le metriche prodotte per ogni sessione sono le seguenti:

- La **Coverage raggiunta su una singola pagina**, ottenuta come rapporto fra il numero di *highlighted widget* e quello di *total widget* della pagina.

  Si noti che le pagine di un'applicazione sono spesso dinamiche e il loro contenuto si modifica nel tempo, una coverage può quindi variare anche repentinamente se vengono rimossi o aggiunti elementi.

- La **Coverage totale della sessione**, ottenuta calcolando la media delle coverage raggiunte nelle singole pagine che compongono la sessione.

  Se una pagina visitata non ha subìto nessuna interazione, questa viene esclusa dal computo della coverage per evitare di alterare la reale coverage della sessione con pagine delle quali non si ha interesse.

- Il **Tempo medio per interazione**, ottenuto calcolando il rapporto tra la durata in secondi della sessione e il totale delle interazioni avvenute in ogni pagina.

- Il **Numero medio di interazioni per pagina**, ottenuto calcolando il rapporto tra il totale delle interazioni avvenute su tutte le pagine e il numero di pagine visitate.

- Il **Numero di pagine scoperte per la prima volta**, ossia il numero di pagine che non erano mai state visitate fino a quel momento in una determinata sessione.

  Per calcolare questo numero viene conservato l'elenco delle pagine visitate (esplicitandone per ognuna lo stato) in modo permanente nel database interno dell'applicazione e, al termine della sessione, viene fatta la differenza insiemistica tra le pagine visitate nella presente sessione e quelle già visitate in precedenza.

- Il **Numero di widget che sono stati scoperti per la prima volta**. In questo caso vengono memorizzati in modo permanente tutti i widget su cui sia avvenuta un'interazione per ogni pagina. Man mano che il tester agisce sui widget durante la sessione, si controlla se in tale pagina si fosse già verificata un'interazione con quel particolare widget.

- Il **Numero di issue** riportati durante la sessione. Gli issue sono i problemi verificatisi nel sistema, riconosciuti dal tester e segnalati, tipicamente con un commento.

- Il **Numero di easter egg** costituisce una metrica che, seppur piuttosto inusuale, indica con quanta profondità è stato esplorato il dominio; tale valore è un numero assoluto ed è intrinsecamente inversamente proporzionale al numero di collegamenti che sussistono tra le pagine.

  Considerando il SUT come un grafo orientato non pesato, le varie pagine che costituiscono l'applicazione sono i nodi del grafo e i collegamenti tra le varie pagine sono gli archi; per ogni nodo esiste un solo arco che, se percorso, contiene l'easter egg. Aumentando i collegamenti tra le pagine (gli archi del grafo) diminuisce la probabilità di percorrere esattamente l'arco che porti alla generazione dell'easter egg.

2.6 Adattamento del Tool

Come si evince dalla Figura 2.1, il principale punto di collegamento tra Scout e i moduli che implementano la Gamification è la classe Java **AppiumPlugin**, la classe che contiene i riferimenti alle Appium Client Libraries.

Questa classe, che da principio aveva il ruolo di legante tra la logica degli stati di Scout e le invocazioni alle funzioni di libreria per effettuare le richieste di Appium, è stata arricchita con la logica di Gamification e di raccolta dei dati.

All'avvio della sessione di *AppiumPlugin*, viene avviato Appium Server in ascolto all'indirizzo IP 0.0.0.0 sulla porta 4723 tramite l'apposito comando di shell; dopo aver messo il thread corrente in attesa dell'inizio dell'operatività del Server, vengono settate le *desired capabilities*, ovvero una mappa contenente tutti i parametri desiderati per avviare la connessione tra l'Appium Server e l'AVD. Se la creazione del driver è andata a buon fine si prosegue la creazione del Model interno e viene infine inizializzata anche la sessione di Gamification con un nuovo albero delle pagine, istanziando un oggetto di classe *Session*.

AppiumPlugin possiede il metodo *verifyAndReplace()* che permette di gestire i cicli di refresh della schermata: in questo metodo vengono controllati i widget in evidenza contenuti nello State Controller che devono essere mostrati nell'AL. In questa fase vengono filtrati i widget che hanno una dimensione non nulla, ma sono effettivamente cliccabili. Quando questo metodo è chiamato per la prima volta su una pagina nuova si sceglie in modo casuale uno tra i widget cliccabili presenti dotati di ID, il quale, se cliccato, attiva nella pagina un *easter egg* (un elemento ludico spiegato in seguito nell'apposita sezione).

Per ogni interazione verificatasi e captata da *AppiumPlugin*, grazie al Gamification Engine viene calcolata una stringa descrittiva del tipo di azione accaduta, la quale, invocando il metodo *newInteraction()*, viene salvata all'interno della rappresentazione ludicizzata della pagina, controllando se l'interazione con il widget corrispondente sia già avvenuta in passato, oppure sia avvenuta per la prima volta.

La maggior parte delle modifiche apportate a *AppiumPlugin* sta nelle operazioni che si verificano ad ogni tocco su un widget presente nella pagina. Quando si riconosce un tap, oltre a registrare l'interazione avvenuta, si emula il tocco in diversi modi: se il widget è individuato da un ID, una stringa di testo o dalla content description, allora viene riportato il click sull'elemento tramite l'apposita funzione dell'Appium Client Library; se invece l'elemento non può essere identificato da nessuno di questi attributi, viene eseguito il click "manualmente" segnalando direttamente all'AVD le coordinate del punto in cui il tocco è avvenuto tramite il corrispondete comando di shell.

Successivamente si verifica se l'elemento cliccato coincida con quello estratto casualmente per attivare l'easter egg, in tal caso vengono generano le coordinate

e reso visibile tale elemento di Gamification. Dopodiché viene interrotto il timer relativo alla pagina corrente, salvato lo stato di Scout nella pagina corrente e si procede alla determinazione del prossimo stato della applicazione. Si determina lo stato del SUT dopo aver ricevuto ed elaborato il click, dopodiché tale stato viene passato come input al metodo *updateState()* di *Session* che permette di determinare il nuovo nodo *current*.

Se lo stato è rimasto lo stesso, anche la pagina e *current* rimangono inalterati; se lo stato è cambiato assumendo un valore già presente nell'albero, allora *current* assume come valore quel particolare nodo e si ripristina lo stato di Scout; se invece il tocco ha effettivamente condotto ad uno stato dell'applicazione mai visitato, allora si aggiunge il nuovo nodo all'albero della sessione come figlio diretto del precedente nodo. Nel caso in cui lo stato sia effettivamente mutato vengono salvati nel database i dati relativi a widget con i quali nessuno aveva finora interagito e salvata la miglior coverage assoluta della pagina come highscore. Dopo aver sincronizzato lo stato del SUT con quello della sessione di Gamification e con quello di Scout, viene fatto ripartire il timer della pagina appena aperta.

Infine, dopo la chiusura del driver, vengono arrestati i timer attivi, calcolati tutti i dati necessari per la schermata di riepilogo della sessione ludicizzata (mostrata grazie alla classe *RecapGUI*) e infine salvati in modo permanente nel database i dati che riguardano quest'ultima (come highscore delle pagine, widget in cui si ha interagito, file di log dell'utente ecc.).

2.7 Gamification Engine

Le principali funzionalità relative agli elementi di Gamification sono state implementate all'interno del package **Gamification Engine**, rappresentato nel dettaglio in Figura 2.5. Tutte le classi presenti al suo interno hanno il compito di elaborare le metriche relative alla sessione appena conclusa e di renderle persistenti, eventualmente caricando prima in memoria quelle già presenti all'interno del database, qualora il tester avesse già eseguito delle sessioni di testing precedentemente, per aggiornarle alla luce dei nuovi risultati conseguiti. Si noti che, per ragioni di semplicità, si è preferito l'uso di file testuali salvati localmente per memorizzare le informazioni, piuttosto che l'utilizzo di un database.

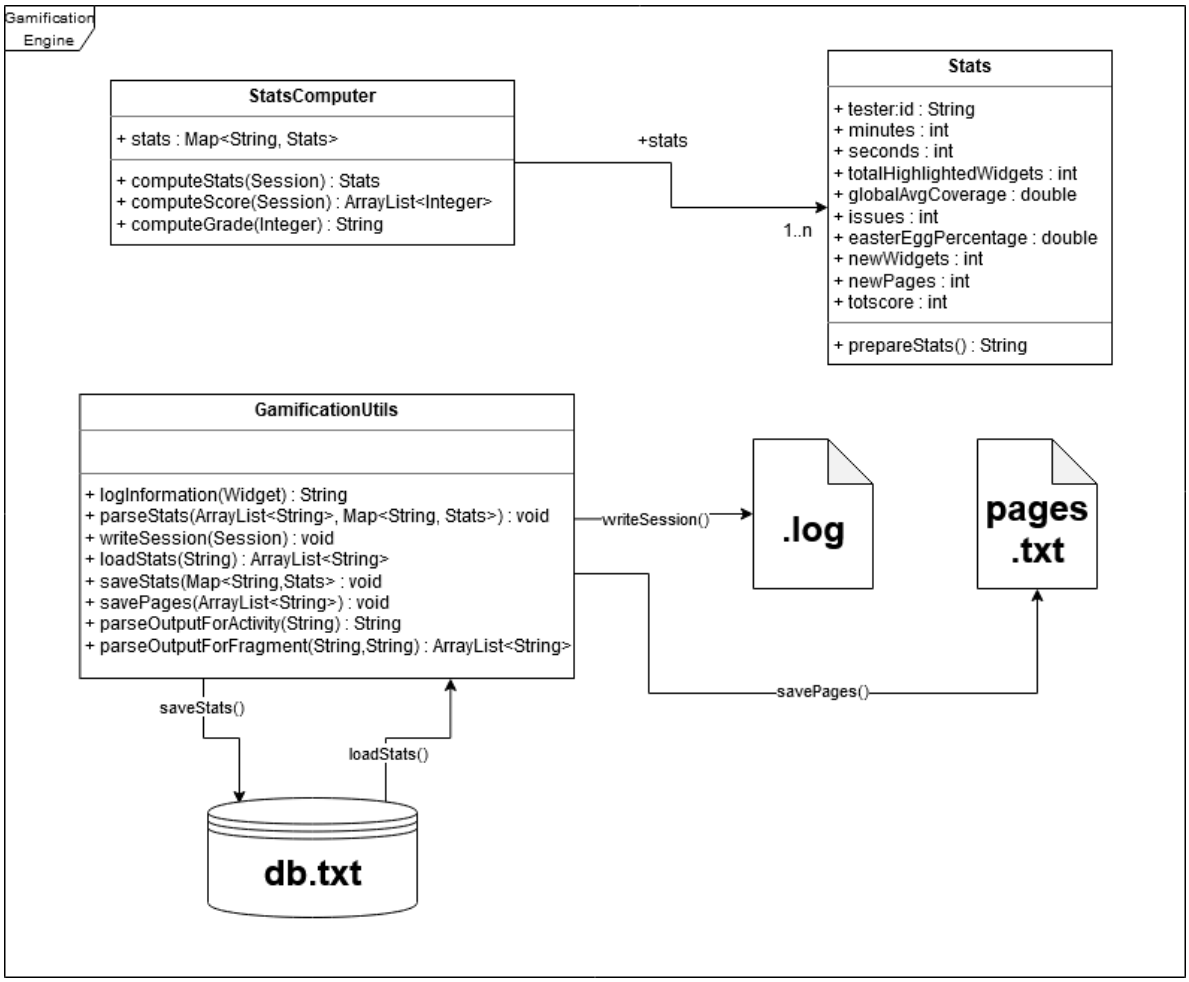

Figura 2.5. Package contenente le classi che implementano le funzionalità relative agli elementi di Gamification.

Le principali funzionalità legate agli elementi di Gamification introdotti dal plugin (spiegati nel dettaglio nella sezione successiva), vengono implementate con l'ausilio di due classi fondamentali e fortemente interconnesse: **StatsComputer** e **Stats**.

Ad ogni tester è associata una serie di statistiche che esso può costantemente aggiornare eseguendo più sessioni di testing all'interno della piattaforma. Tutte queste statistiche vengono salvate come attributi di un oggetto *Stats* associato a quel tester, ognuna delle quali permette di capire l'andamento dell'utente nell'uso del tool. Tali statistiche (legate alle metriche spiegate nella sezione precedente) sono:

- ID del tester;
- Minuti e secondi totali trascorsi svolgendo delle sessioni di testing;
- Numero totale di widget con cui il tester ha interagito nel corso delle varie sessioni;
- Coverage media raggiunta nel corso delle varie sessioni;
- Numero totale di issue segnalati;
- Percentuale di easter egg attivati;
- Numero totale di widget e pagine scoperti dal tester in esame;
- Punteggio totale ottenuto nel corso delle varie sessioni svolte.

All'inizio di ogni sessione, viene istanziata la classe singleton *StastComputer* che mantiene al suo interno una mappa che rispecchia il contenuto del database, associando ad ogni tester ID il corrispettivo oggetto *Stats*. Tale mappa viene creata al momento dell'istanziazione dell'oggetto di classe *StatsComputer*, andando a recuperare dal database (file *db.txt*) le informazioni necessarie, con l'ausilio della classe *GamificationUtils* (spiegata nel dettaglio successivamente). All'interno del database, ogni utente possiede una propria entry corrispondente ai dati del proprio oggetto *Stats*. Si noti che la classe *Stats*, inoltre, fornisce un metodo che permette di organizzare le informazioni contenute negli attributi dell'oggetto in un formato conforme alla sintassi utilizzata all'interno del database. In questo modo, sarà sempre possibile parsificare il file *db.txt* correttamente ad ogni istanziazione di *StatsComputer*.

La principali funzionalità della classe *StatsComputer* consistono nel calcolo delle metriche relative alla sessione corrente (metodo *computeStats()*), nella loro aggregazione con quelle già presenti nella mappa delle statistiche e nel calcolo del punteggio finale e del voto corrispondente (metodi *computeScore()* e *computeGrade()*). In particolare:

- *computeStats()*: metodo che permette di estrapolare le metriche a partire dai dati grezzi relativi all'oggetto *Session* che viene passato come parametro. Questo metodo calcola tutte le statistiche necessarie e controlla se il tester corrente possieda già un'entry all'interno del database (ossia se sia presente

nella mappa *stats*): in caso affermativo, aggiorna l'entry corrispondente all'interno della mappa, altrimenti ne crea una nuova e la aggiunge alla struttura dati.

- *computeScore()*: metodo che si occupa del calcolo delle varie componenti del punteggio finale (spiegato nel dettaglio nell'omonima sezione), sulla base delle metriche relative alla sessione. Il valore di ritorno consiste in una coppia di interi che rappresentano, rispettivamente, il punteggio base e il punteggio bonus totalizzati dal tester.

- *computeGrade()*: metodo che determina il voto da assegnare alla performance del tester, sulla base del punteggio complessivo ottenuto.

Il modulo che implementa il Gamification Engine contiene inoltre una classe di utility chiamata *GamificationUtils*, che fornisce una serie di metodi statici che consentono di eseguire diverse operazioni di lettura e scrittura di dati sui file che permettono di rendere persistenti le informazioni relative alle varie sessioni.

Le principali funzionalità messe a disposizione dalla classe sono:

- Creazione di una stringa che descriva un'interazione appena avvenuta, in base al tipo di widget con cui il tester ha interagito. Ciò è reso possibile dal metodo *logInformation()*. Tutte le interazioni così codificate vengono inserite all'interno di un array list dell'oggetto *Page* corrispondente, una volta terminata la sessione (quindi all'interno del metodo *stopSession()* di *AppiumPlugin*), con una chiamata del metodo *writeSession()*. Tale metodo va a creare un file di log (il cui nome viene generato a partire dall'ID del tester e dal timestamp relativo alla conclusione della sessione) che viene salvato all'interno di una cartella avente come nome l'ID del tester, posizionata nella directory */Gamification*.

- Caricamento di dati e statistiche. Quest'operazione è resa possibile dal metodo *loadStats()*, che permette di leggere le informazioni contenute in un file il cui path viene passato al metodo come argomento. *loadStats()* viene inoltre invocato nell'ambito dell'istanziazione della classe singleton *StatsComputer*, dove viene utilizzato per caricare in memoria il database contenente le statistiche di tutti i tester che hanno già compiuto almeno una sessione di testing fino a questo momento (file *db.txt*). Tali informazioni, una volta caricate in

memoria, vengono parsificate tramite il metodo *parseStats()*, che permette di creare un oggetto *Stats* per ogni utente presente nel database e di associarlo all'ID del tester a cui fa riferimento, rendendo possibile la creazione di una mappa che viene salvata nell'attributo corrispondente all'interno dell'istanza corrente di *StatsComputer*. Infine, *loadStats()* viene utilizzato anche per caricare in memoria, una volta tantum, l'elenco delle pagine già visitate in precedenza da almeno un tester che abbia completato una o più sessioni prima di quella corrente. Tale operazione viene eseguita invocando il metodo all'interno del costruttore dell'oggetto *Session* relativo alla sessione corrente, passando come parametro il path del file *pages.txt*.

- Salvataggio delle statistiche in memoria. Quest'operazione viene eseguita al termine della sessione grazie al metodo *saveStats()*, una volta che *StatsComputer* ha terminato di aggiornare la mappa delle statistiche alla luce dei risultati ottenuti dal tester che ha completato la sessione corrente.

- Salvataggio dell'elenco contenente le pagine visitate da tutti i tester. Quest'operazione viene eseguita, dopo una serie di chiamate, all'interno di un metodo di *StatsComputer*, utilizzando il metodo statico *savePages()* di *GamificationUtils* e permette di rendere persistente l'elenco aggiornato delle pagine esplorate da tutti i tester che hanno eseguito almeno una sessione di testing (inclusa quella corrente, appena terminata).

- Parsing dell'output del comando di shell per recuperare le informazioni sull'Activity in esecuzione e sui Fragment in essa attivi rispettivamente con i metodi *parseOutputForFragment()* e *parseOutputForActivity()* che, utilizzati insieme, permettono di ottenere lo stato della pagina attiva sotto forma di array list contenente la collezione dei Fragment e Activity correnti.

2.8 Elementi di Gamification applicati

Durante la selezione ed implementazione delle metriche sono state effettuate numerose sessioni di prova su alcuni domini ben noti. Le metriche ottenute durante queste sessioni hanno messo in luce alcuni elementi di Gamification consoni, tra cui alcuni applicati fin da subito, alcuni solo teorizzati e di seguito riportati per eventuali sviluppi futuri.

2.8.1 Barra di progresso

Il primo elemento applicato è stata una barra di progresso che mostrasse visivamente il grado di completezza raggiunto dal tester nella pagina corrente.

In ogni pagina viene mostrata nella parte superiore della schermata una sottile linea rossa che si adatta in modo da occupare tutta l'ampiezza della schermata; tale linea rappresenta la quantità di widget rimanente in percentuale per ottenere una page coverage del 100%. Una linea verde sovrasta quella rossa per rappresentare la coverage percentuale raggiunta in un determinato momento nella pagina come si evince in Figura 2.6.

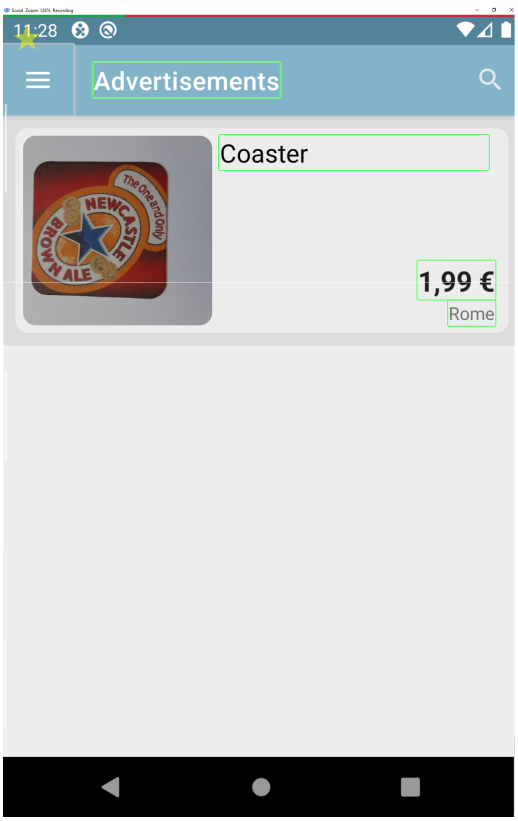

Figura 2.6. Barra di progresso di una sessione di testing nell'app aMADzon2

Ad ogni interazione su un nuovo widget registrata, la barra di avanzamento viene aggiornata in tempo reale: ciò permette al tester di capire quanto sia profonda l'esplorazione della pagina in esame e di intuire spannometricamente quanti widget interagibili siano presenti nel nodo corrente.

Ad integrare la coverage corrente, la barra di progresso mostra inoltre la miglior coverage mai raggiunta nella pagina, visualizzata con una linea blu che si interpone tra quella verde e quella rossa: ciò permette al tester di dedurre quanto la pagina fosse stata esplorata in precedenza da terzi. Se nella sessione la coverage è quella massima mai ottenuta, la linea verde sovrasta in estensione quella blu.

Le informazioni che si traggono da tale elemento ludico sono solamente quantitative, non fornisce nessuna informazione riguardante collocazione e tipologia dei widget. Gli obiettivi di tale elemento sono quelli di rendere consapevole il tester dei progressi ottenuti sulla pagina corrente, di trasmettere un senso di soddisfazione al raggiungimento della coverage massima e infine di suscitare nel tester un lieve senso di sfida nei confronti dei tester passati. La barra di progresso nella sua interezza è mostrata in Figura 2.7.

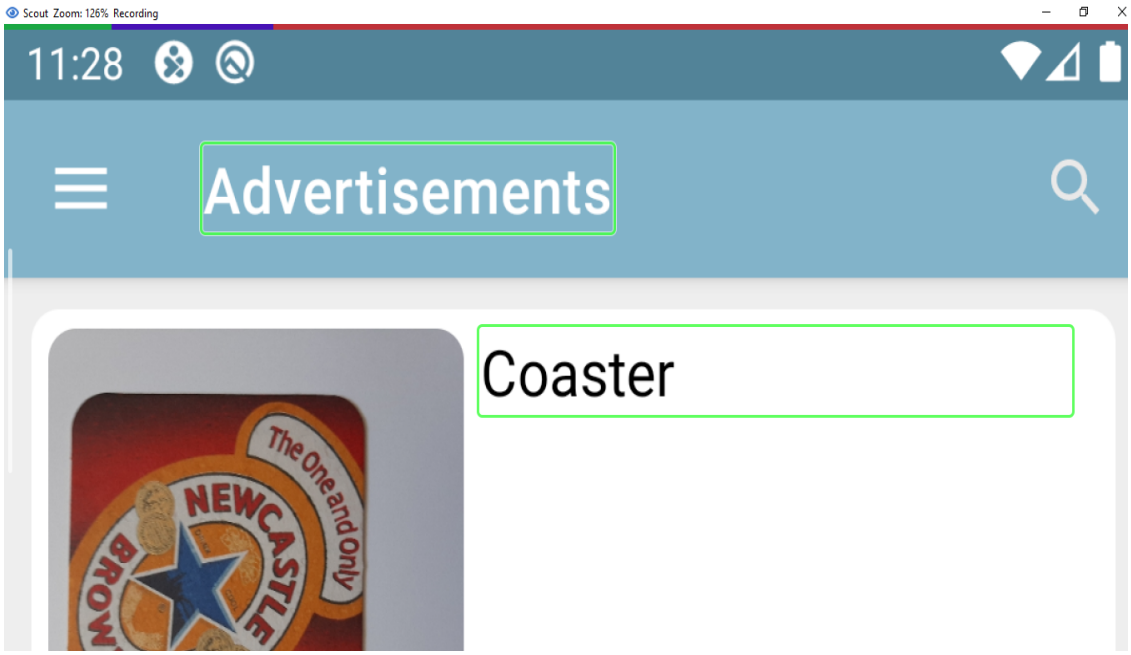

Figura 2.7. Barra di progresso che mostra la coverage corrente in verde e la coverage massima raggiunte sulla pagina in blu.

2.8.2 Punteggio

Al termine di una sessione di testing, al tester viene mostrata una schermata che presenta i risultati ottenuti durante la sessione stessa. Vengono dunque mostrate le

metriche principali (descritte nella sezione precedente) oltre ad un **punteggio** che riassume la performance del tester.

Il punteggio viene calcolato prendendo in considerazione diverse metriche e combinandole in modo da ottenere 3 componenti principali e 2 componenti bonus, secondo la seguente formula:

$$\mathbf{P} = \mathbf{a} \cdot \mathbf{C} + \mathbf{b} \cdot \mathbf{EX} + \mathbf{c} \cdot \mathbf{EF} + [\mathbf{d} \cdot \mathbf{T}] + [\mathbf{e} \cdot \mathbf{PR}] \tag{2.1}$$

Le prime tre componenti costituiscono il **punteggio base** e ognuna di esse è pesata con il proprio coefficiente. In particolare:

- **C**: rappresenta la componente di **coverage** e descrive il grado di copertura medio raggiunto dal tester nelle varie pagine visitate durante la sessione, calcolata in termini di widget interagiti sul totale di widget interagibili. In particolare, C viene calcolata come:

$$C = \frac{\sum_{\forall i \in P} pc_i}{|P|} \tag{2.2}$$

  dove al numeratore troviamo la sommatoria delle coverage (pc_i) raggiunte nelle singole pagine dell'insieme P e al denominatore la cardinalità dell'insieme stesso. La coverage relativa ad una singola pagina viene a sua volta calcolata con il seguente rapporto:

$$pc_i = \frac{w_{hl,i}}{w_{tot,i}} \tag{2.3}$$

  dove $w_{hl,i}$ è il numero di widget diversi con cui il tester ha interagito all'interno dell'i-esima pagina del set P, mentre $w_{tot,i}$ è il numero totale di widget cliccabili presenti nella medesima pagina i.

- **EX**: rappresenta la componente **esplorativa** della sessione e si basa sul numero di pagine che il tester considerato ha visitato e che non erano ancora state visitate da nessun altro fino a quel momento. La componente, inoltre, prende anche in considerazione i widget con cui il tester ha interagito per primo, anche all'interno di pagine già visitate in precedenza da almeno un collega. EX_{comp}, nel dettaglio, viene calcolata come:

$$EX = \frac{k}{b} \cdot \frac{p_{new}}{p_{tot}} + \frac{h}{b} \cdot \frac{hw_{new}}{hw_{tot}}, \qquad k + h = b; \tag{2.4}$$

dove il primo rapporto rappresenta la sottocomponente relativa alle nuove pagine scoperte (percentuale delle pagine nuove rispetto al totale delle pagine visitate durante la sessione), mentre il secondo quella relativa ai widget (percentuale degli *highlighted widget* nuovi rispetto al totale dei widget con cui il tester ha interagito nel corso della sessione). h e k rappresentano le percentuali di ogni sottocomponente rispetto al punteggio base totale.

Si noti che il database relativo alle pagine già visitate da un qualsiasi tester è completamente configurabile, perciò nel caso di un esperimento condotto in un ambiente controllato è possibile inserire manualmente alcune pagine del dominio da testare prima che comincino le sessioni di test, ad esempio per evitare che il primo tester ottenga un vantaggio sostanziale (in termini di punteggio) rispetto ai suoi successori.

- **EF**: rappresenta la componente di **efficienza** e serve sostanzialmente a premiare i tester che, a parità di punteggio nelle altre componenti, hanno evitato di ripetere click su widget già evidenziati in precedenza, rispetto a coloro i quali hanno cliccato più volte sui medesimi elementi. Viene dunque calcolata come:

$$EF = \frac{w_{hl}}{w_{int}} \tag{2.5}$$

dove w_{hl} è il numero totale di *highlighted widget* della sessione mentre w_{int} è il numero di interazioni effettuate sui widget (comprensivo quindi dei click effettuati su widget già evidenziati).

Per la natura stessa di questa componente, se il test viene effettuato coscienziosamente, il suo valore sarà tendenzialmente molto vicino a 1.

La somma di queste tre componenti, moltiplicate per i rispettivi coefficienti, permette al tester di incrementare il proprio punteggio fino ad un massimo di 100 punti. Inoltre, per aiutarlo nella comprensione della qualità della propria performance, valutata sulla base del punteggio ottenuto, il plugin fornisce anche un **voto** letterale secondo un meccanismo comunemente utilizzato in molti videogiochi, seguendo una scala con voti che vanno dalla D (voto più basso) alla S (voto più alto), secondo la suddivisione mostrata in Tabella 2.1.

Un tester ha la possibilità di accumulare anche un **punteggio bonus**, che può arrivare al massimo fino al 50% del punteggio base ottenuto. Il punteggio bonus viene sommato a quello base per determinare la fascia di voto del tester,

Tabella 2.1. Voti basati sul punteggio finale

Minimo	Massimo	Voto
100	-	S
80	99	A
70	79	B
50	69	C
0	49	D

permettendo quindi di incrementare il voto che si sarebbe ottenuto con il solo punteggio base. In certi casi, dunque, il punteggio bonus permette anche di superare i 100 punti massimi ottenibili grazie al punteggio base.

Il punteggio bonus si ottiene con la formula:

$$\mathbf{P_{bonus}} = [\mathbf{d} \cdot \mathbf{T}] + [\mathbf{e} \cdot \mathbf{PR}] \tag{2.6}$$

dove:

- **T**: rappresenta la componente **temporale** legata alla durata della sessione. In generale, il plugin vuole premiare chi effettua sessioni più lunghe e quindi (teoricamente) più approfondite, ma per escludere casi limite in cui un tester ha volutamente trascorso troppo tempo sulle pagine per aumentare il proprio punteggio oppure per individuare chi ha cliccato troppo velocemente sui widget soltanto per incrementare il proprio numero di interazioni, viene preso in considerazione anche il tempo medio impiegato per ciascuna interazione. La componente viene dunque calcolata come:

$$T = \begin{cases} 0 & s_{int} \leq 2 \ \vee \ s_{int} > 30 \\ 1.5 \cdot t & 2 < s_{int} \leq 5 \\ t & 5 < s_{int} \leq 15 \\ 0.5 \cdot t & 15 < s_{int} \leq 30 \end{cases} \tag{2.7}$$

  dove t è la durata della sessione calcolata in minuti e s_{int} i secondi medi per interazione.

- **PR**: rappresenta la componente legata alle **problematiche** trovate dal tester

nel corso della sessione. Viene calcolata con la seguente formula:

$$PR = \frac{issue + easteregg}{e} \tag{2.8}$$

$$issue = \begin{cases} y & n_i \geq \frac{y}{\beta} \\ \beta \cdot n_i & n_i < \frac{y}{\beta} \end{cases} \tag{2.9}$$

$$easteregg = \begin{cases} z & n_{ee} \geq \frac{z}{\gamma} \\ \gamma \cdot n_{ee} & n_{ee} < \frac{z}{\gamma} \end{cases} \tag{2.10}$$

$$\text{con} \quad y + z = e$$

dove *issue* ed *easter egg* sono le funzioni per il calcolo delle corrispondenti sottocomponenti. Per ognuna di esse sono definiti dei coefficienti di saturazione: valori massimi che possono assumere le funzioni (rispettivamente y e z). Oltre ai coefficienti di saturazione, vengono definiti anche i coefficienti β e γ che rappresentano il peso che ogni unità di ciascuna problematica ha rispetto alla propria sottocomponente. I restanti termini, invece, rappresentano il numero assoluto di ciascun problema: n_i è il numero di issue aperte dal tester e n_{ee} quello di easter egg attivati.

Sebbene i pesi di questi addendi, come tutti gli altri coefficienti presenti nelle varie componenti del punteggio, siano interamente personalizzabili, è importante dare più rilevanza alla componente legata agli easter egg, poiché tale sottocomponente in molti casi si rivela un buon indicatore della profondità con cui è stata svolta la sessione di testing dall'utente corrente. Per tale motivo, di default, al coefficiente γ viene assegnato il quadruplo del valore di β.

Questa componente viene inclusa nel punteggio bonus anziché in quello base poiché potrebbero verificarsi dei casi in cui l'interfaccia soggetta al test non presenti alcuna problematica e ciò non dovrebbe inficiare in nessun modo la performance del tester.

Le componenti appena descritte vengono moltiplicate per i rispettivi coefficienti *a, b, c, d* ed *e*, tutti interamente modificabili tramite l'apposito file di configurazione. Nonostante ciò, vengono proposti i seguenti valori di default, che vengono inoltre attribuiti ai coefficienti nel caso in cui i valori presenti nel file di configurazione non rispettassero i vincoli:

- $a = 0.6$;
- $b = k + h = 0.1 + 0.2 = 0.3$;
- $c = 0.1$;
- $d = 0.25$;
- $e = y + z = 0.05 + 0.20 = 0.25$;

Al termine di una sequenza di testing, il plugin mostra al tester i dati relativi alla sessione appena conclusa all'interno di una finestra di recap, illustrata in Figura 2.8. Tale schermata mostra un riepilogo strutturato delle metriche più importanti che contribuiscono al calcolo del punteggio finale, anch'esso indicato al centro della finestra, distinguendo fra punteggio base e punteggio bonus. Al fondo del riepilogo, inoltre, viene mostrato il voto assegnato alla sessione, sulla base del punteggio totale ottenuto dal tester.

Tramite l'apposito tasto presente al fondo della schermata di riepilogo, il tester ha inoltre la possibilità di visualizzare lo storico delle proprie statistiche complessive, aggiornato alla sessione appena conclusa. Tale schermata, rappresentata in Figura 2.9, mostra sostanzialmente l'entry del database relativa al tester che ha svolto la sessione di testing appena conclusa, aggregando tutte le metriche delle sessioni da lui effettuate precedentemente. Tale entry consiste in un oggetto di classe *Stats* ritornato dal metodo *computeStats()* di *StatsComputer*, che viene parsificato e passato a sua volta come parametro ad una classe specifica che produce la GUI della schermata di riepilogo.

2.8.3 Classifica

Al termine di una sessione, il tester, premendo l'apposito tasto al fondo della schermata di riepilogo, ha la possibilità di visualizzare una classifica degli utenti che hanno totalizzato il maggior numero di punti nel corso delle varie sessioni. Tale

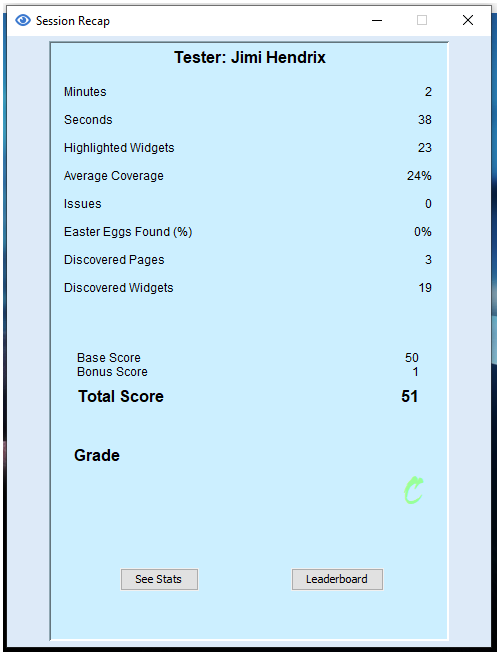

Figura 2.8. Cattura della schermata di riepilogo, contenente tutte le metriche relative alla sessione appena conclusa.

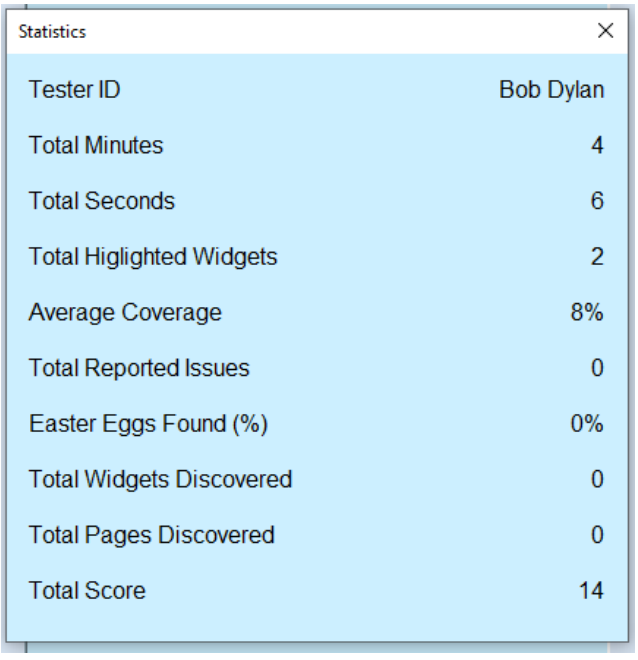

Figura 2.9. Cattura della schermata contenente lo storico relativo ad un tester.

classifica, mostrata in Figura 2.10, mostra i dieci migliori tester che hanno effettuato almeno una sessione di testing, ordinati per punteggio complessivo.

Questo elemento di Gamification è ritenuto particolarmente importante poiché ha come obiettivo l'introduzione di una componente competitiva all'interno del tool: ogni tester, infatti, potrebbe sentirsi motivato ad effettuare sessioni migliori per accumulare più punti possibili e superare i propri colleghi in classifica.

L'introduzione di questo elemento apre la strada a scenari interessanti nell'ambito di sessioni di testing svolte in un contesto controllato: per esempio si può stabilire un certo periodo di validità della classifica e premiare, con una ricompensa reale, il tester che al termine del periodo prestabilito ha totalizzato il maggior numero di punti. Inoltre, andando per esempio a limitare il numero di sessioni effettuabili da ciascun tester, la classifica potrebbe portare ad un incremento dell'efficienza dei tester, incentivandoli a svolgere sessioni di maggiore qualità.

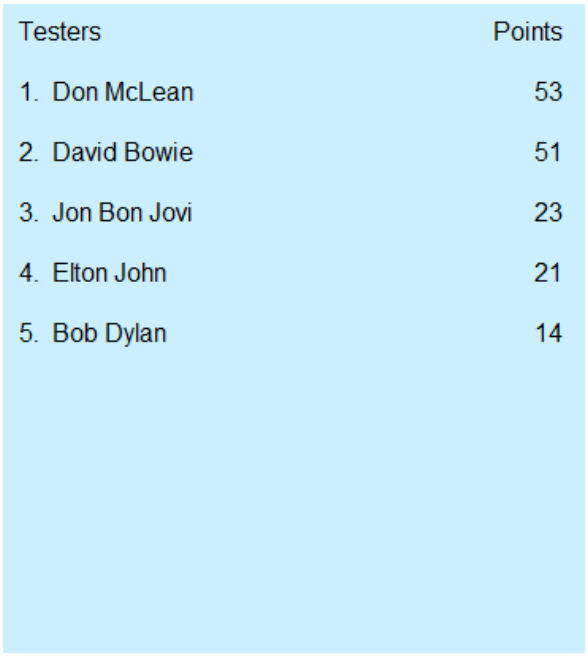

Figura 2.10. Cattura della schermata contenente la classifica dei cinque tester che hanno totalizzato il punteggio massimo nel corso delle varie sessioni di testing.

2.8.4 Easter Egg

In una sessione di testing su un dominio noto, si deve verificare che l'applicazione risponda agli input ricevuti modificando il proprio stato interno in modo consono

e producendo un particolare output. In un'app Android è necessario testare tutti gli elementi di input e di output che compongono la pagina. Non è infrequente che alcuni siano interconnessi tra di loro, richiedendo particolari condizioni di validità che non sempre sono esposte in modo chiaro all'utente; l'app deve perciò eseguire molti controlli sugli input ricevuti, considerando le molte condizioni di utilizzo.

Il plugin di Gamification introduce il concetto di easter egg come un elemento visivo esterno all'applicazione di forma di ovale e dimensioni contenute (inscritto in un rettangolo di 30x50 pixel), presente in sovraimpressione nella pagina dopo aver interagito su un elemento estratto casualmente al fine di simulare un uso scorretto dei widget da parte dell'utente. Un esempio di easter egg attivato in una pagina si può vedere in Figura 2.11.

Ogni volta che viene aperta una pagina, a partire dalla radice, viene estratto a sorte uno tra gli elementi cliccabili dotati di ID presenti nella pagina. Se tale elemento viene cliccato, l'Augmented layer inserisce l'ovale colorato in un punto casuale. Per massimizzare la probabilità di trovare l'easter egg, il tester deve esplorare il più possibile l'app, cliccando sul maggior numero possibile di oggetti nella schermata alla ricerca di quello che funge da attivatore.

L'inserimento di tale elemento è finalizzata ad invogliare il testo a condurre una sessione di testing con una coverage media più alta, proprio perché l'operazione di ricerca dell'easter egg presuppone la produzione un numero maggiore di interazioni su widget diversi. L'obiettivo mira, inoltre, ad indurre il tester ad effettuare interazioni con il maggior numero di widget diversi per riuscire ad attivare l'easter egg e a farlo non seguendo necessariamente l'ordine prestabilito formalmente dal programmatore.

2.8.5 Evidenziazione delle pagine nuove

Tra gli elementi visivi introdotti nel plugin, ne è stato introdotto anche uno che servisse a testimoniare l'abilità del tester in una particolare sessione. Tale elemento consiste nel marcare una pagina dell'app che non era mai stata scoperta fino a quel momento con una stella semitrasparente in alto a sinistra.

A differenza dei badge e delle ricompense che vengono spesso utilizzate per testimoniare il raggiungimento di un obiettivo o certificare l'abilità del tester e sono dei riconoscimenti persistenti che, una volta raggiunti non possono essere revocati,

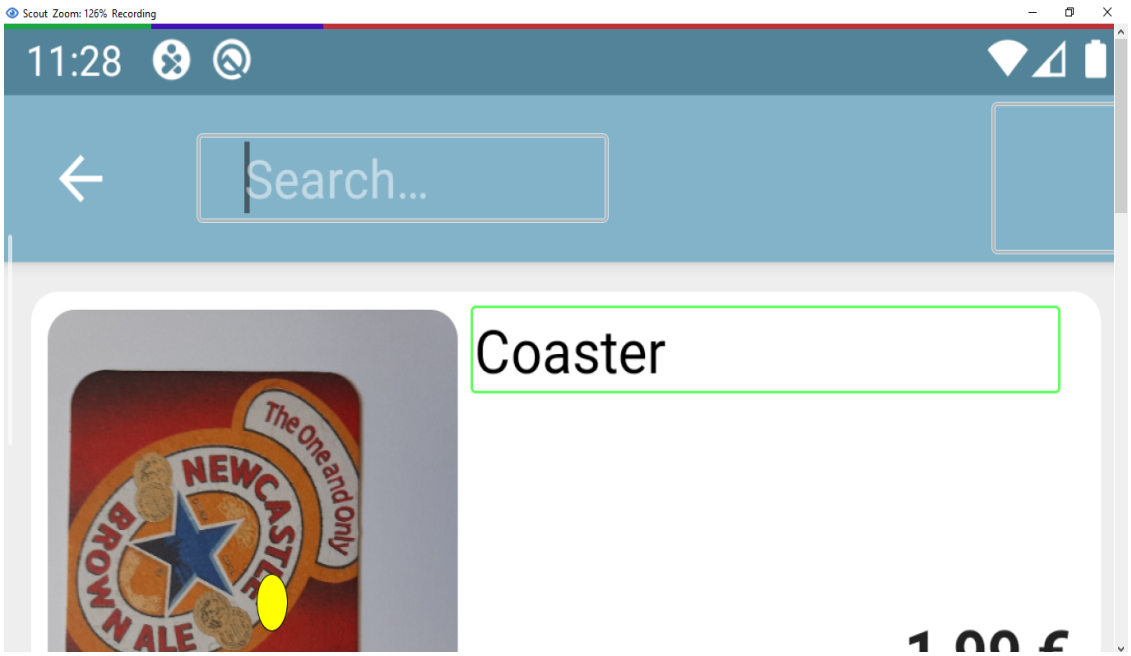

Figura 2.11. Cattura della schermata in cui è presente l'easter egg rappresentato come un ovale giallo.

tale elemento è una via di mezzo tra un premio ed un feedback visivo, infatti al termine della sessione non sarà più visibile al tester.

2.8.6 Elementi teorizzati

Un possibile elemento di Gamification applicabile nel plugin consiste nell'introduzione di meccanismi di sfida. Tale elemento è solo teorizzato e lasciato a possibili ulteriori sviluppi della presente tesi. Le sfide si dividono in due tipi: le sfide esplorative su un domino e le sfide di velocità su una pagina o un insieme di pagine.

Le sfide esplorative consistono nella ricerca del maggior numero di easter egg possibili e nella visita del maggior numero di pagine possibili. Il tester potrà selezionare un'app da utilizzare come oggetto di sfida tra un pool di app esistenti da testare o inserirne uno a sua scelta. In questo caso è necessario che tale applicazione venga corredata con il file APK di installazione e vengano correttamente configurati il nome dell'Activity principale che permette di lanciare l'app e il package in cui localizzarla.

Una volta selezionato il dominio di testing, il tester avrà la libertà di esplorare quante più pagine possibile senza essere cronometrato. Per vincere la sfida dunque

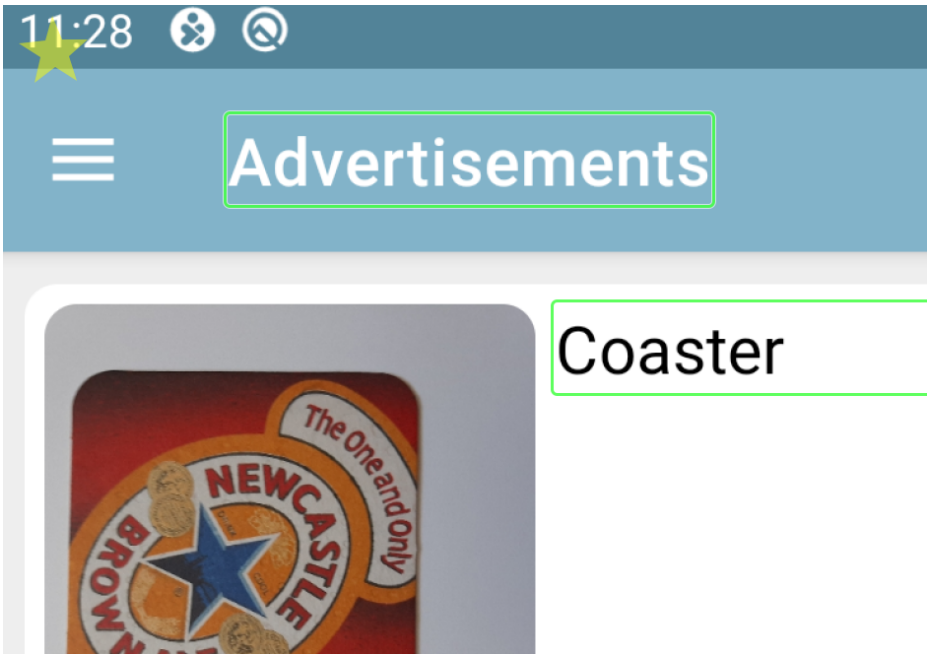

Figura 2.12. Cattura di una pagina mai stata visitata, contrassegnata dalla stella gialla in trasparenza.

bisognerà superare l'attuale record di pagine esplorate per applicazione o di eguagliare il record, ma attivando un numero maggiore di easter egg durante la sessione di testing.

Le sfide di velocità su un insieme di pagine, invece, consistono nell'ottenimento della massima coverage di sessione possibile sull'insieme prestabilito. Il tester partecipa alla sfida scegliendo nuovamente l'app tra un pool di possibilità esistenti o aggiungendone una al pool seguendo gli stessi criteri della sfida esplorativa.

Il tester dovrà quindi essere il più veloce ad interagire su tutti i widget nel minor tempo possibile; a parità di coverage sarà infatti il miglior tempo a decretare il vincitore della sfida.

Quando un tester risulta essere il campione di un dominio su cui è attiva una o più sfide, viene mostrato il trofeo di vittoria ogni volta che si esegue una sessione di testing su quel particolare dominio. Entrambe le sfide possono essere facilmente resettate, nel caso in cui ad esempio il dominio di testing venisse aggiornato ad una nuova release o su base periodica (ogni trimestre/semestre).

Un altro possibile elemento di Gamification consiste nell'implementazione degli

achievement, ossia di particolari riconoscimenti permanenti che ogni tester può guadagnare una volta raggiunti determinati obiettivi.

Ogni achievement generalmente ha un nome particolare che descrive, spesso anche simpaticamente, l'obiettivo da raggiungere (che può anche essere nascosto, nel caso degli achievement più rari), ossia la missione da compiere per ottenere l'achievement stesso. Una volta conquistato un achievement, esso sarà visualizzabile permanentemente nell'apposita sezione del profilo del tester, dove sarà quindi possibile tenere traccia degli achievement ottenuti e di quelli non ancora guadagnati.

Solitamente gli achievement sono uguali per tutti i tester che utilizzano il tool e sono in numero prestabilito. Ciò comporta che sia possibile realizzare una graduatoria degli utenti in base al numero di achievement collezionati, oltre al fatto di poter classificare gli achievement stessi con una scala di rarità in base alla percentuale di utenti che hanno raggiunto ciascun obiettivo.

Ci possono essere diverse tipologie di achievement, spaziando da quelli introduttivi (es. "Chi ben comincia..": "Hai effettuato la tua prima sessione di testing."), che cercano di fidelizzare gli utenti ricompensandoli fin dalle prime fasi di utilizzo della piattaforma così da invogliarli a tornare, fino a quelli cumulativi, più comuni, che premiano particolari traguardi come la ripetizione di una determinata operazione un certo numero di volte (es. Achievement "Stakanovista!": "Hai effettuato 100 sessioni di testing."). Altri ancora, invece possono corrispondere a piccole sfide isolate (es. "Gallina dalle uova d'oro": "Hai trovato tutti gli easter egg presenti in una singola sessione di testing."), con cui si vuole premiare un tester che riesca a far accadere un certo evento in un dato intervallo di tempo.

Gli achievement sono strettamente legati ad un altro elemento di Gamification facilmente implementabile, ossia i badge. I badge non sono altro che coccarde virtuali ottenibili da ciascun tester al raggiungimento di determinati obiettivi e ciascuno di essi certifica una particolare competenza acquisita dal tester.

I badge sono dunque legati ad obiettivi meno specifici rispetto a quelli a cui fanno riferimento gli achievement, sebbene siano anch'essi legati al raggiungimento di determinati traguardi. Presentano inoltre altre due differenze sostanziali:

1. Ogni badge corrisponde ad una coccarda con un aspetto unico, visualizzabile immediatamente all'interno del profilo di ciascun tester. Ciò può spingere

gli utilizzatori del plugin a voler collezionare più badge possibili anche per migliorare esteticamente il proprio profilo personale.

2. Dato che ogni badge certifica una determinata competenza acquisita dal tester, è possibile esportarli in un file esterno al plugin in modo da poter attestare, anche al di fuori del contesto del tool, l'avvenuta acquisizione di tali conoscenze.

Capitolo 3

Esempio di Sessione

Grazie alle classi descritte nel dettaglio nella sezione 2.4, il plugin permette di creare una rappresentazione della sessione di testing effettuata dal tester corrente, creando un albero delle pagine visitate.

Per illustrare il funzionamento di questo meccanismo di rappresentazione, si è scelto di effettuare una semplice sessione di testing sull'applicazione di compravendita tra privati aMADzon2 (Cacciotto, Fulcini, Piacentini, & Pietrasanta, 2020), che si presta adeguatamente alla realizzazione di un esempio di utilizzo del plugin per via della semplicità dei meccanismi che gestiscono le pagine.

L'applicazione, di default, richiede la registrazione dell'utente o l'autenticazione dello stesso tramite un account Google. Si noti che questa procedura di registrazione e di autenticazione è stata fatta in precedenza, mantenendo le informazioni corrispondenti nella cache dell'AVD utilizzato. Inoltre, al fine di simulare un utilizzo prossimo a quello di regime, è stato inserito un oggetto messo in vendita, così da permettere all'utente di interagire con esso. Per questi motivi lo stato di partenza non è nullo e gli step seguenti sono legati allo stato descritto, che può essere facilmente ricostruito in caso si parta da uno stato ex novo.

La sessione è stata svolta seguendo i passi qui elencati:

1. All'interno della pagina iniziale dell'app, contraddistinta dallo stato [.MainActivity, OnSaleListFragment], viene mostrata una *RecyclerView* contenente una lista di oggetti, tra questi ne è stato cliccato uno;

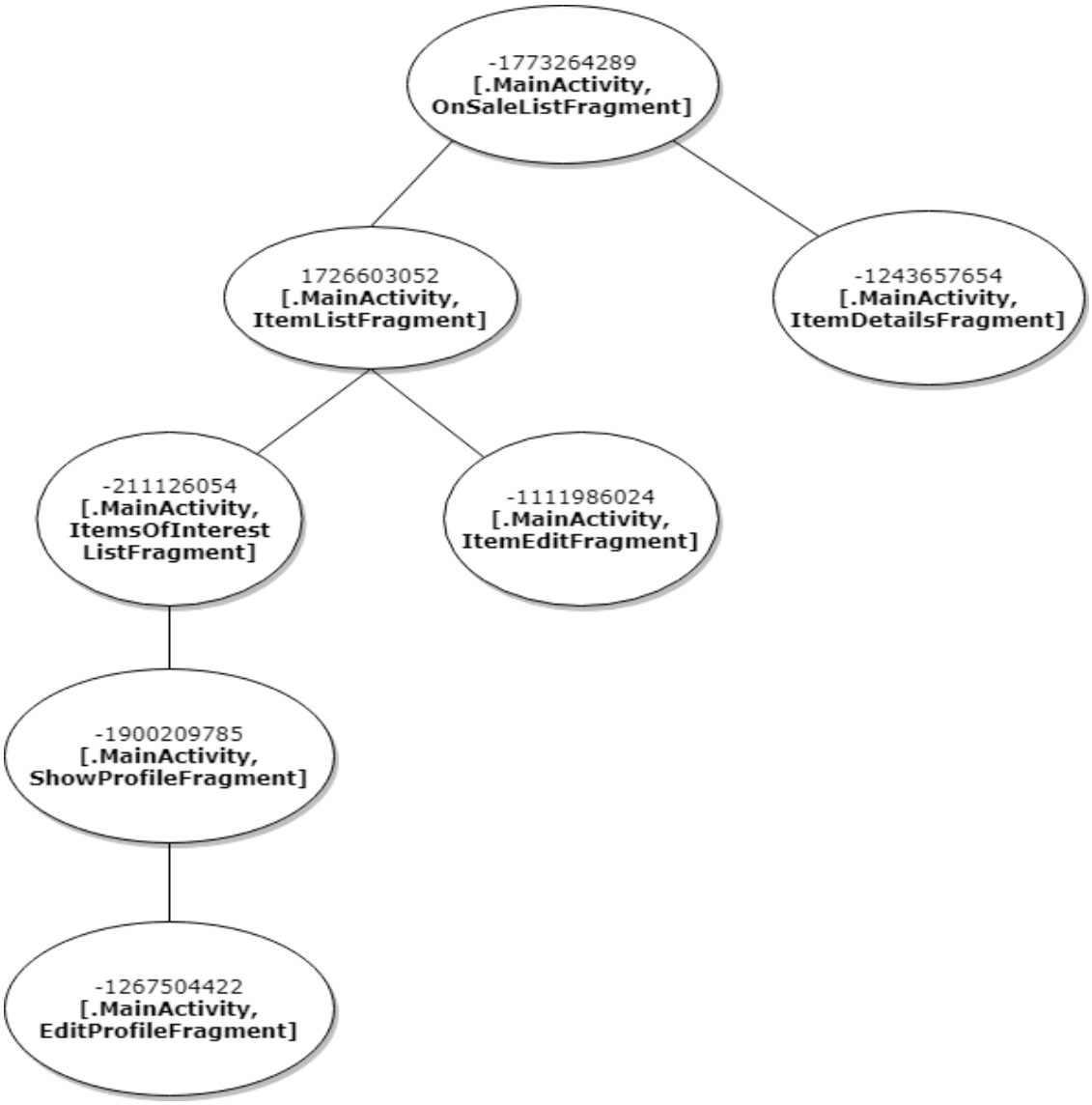

Figura 3.1. Rappresentazione grafica dell'albero della sessione.

2. Il tocco sull'oggetto ha portato ad una schermata ([.MainActivity, ItemDetailsFragment]) in cui sono mostrati i dettagli dell'oggetto in vendita. In basso a destra è presente un Floating Action Button (FAB) con una fiamma, necessario a esprimere interesse nell'oggetto in esame. Questo pulsante è stato cliccato ed è stato mostrato un feedback visivo sull'azione avvenuta, dopodiché si è premuto il tasto "Indietro" che porta alla pagina precedente;

3. Tornati alla pagina iniziale, è stato aperto il Navigation Drawer con l'icona

in alto a sinistra e da questo menù è stata selezionata l'opzione "My Items";

4. La nuova pagina ha portato ad una pagina il cui stato è [.MainActivity, ItemListFragment]. La pagina si presenta come una pagina vuota in cui viene segnalato all'utente l'assenza di oggetti in vendita da parte sua. In basso a destra è stato cliccato il FAB, con un aspetto diverso da quello precedente, che permette di aggiungere un annuncio;

5. A seguito del tocco sul FAB viene aperta una schermata nuova ([.MainActivity, ItemEditFragment]) che permette di inserire i dati necessari alla creazione dell'annuncio; dopo l'inserimento di questi dati si procede al salvataggio tramite l'apposito pulsante.

6. Il salvataggio dell'oggetto ha portato alla schermata visitata in precedenza, con la differenza che il messaggio circa l'assenza di oggetti ha lasciato spazio ad una *RecyclerView* contenente l'annuncio creato con le informazioni inserite nello step precedente. È stato nuovamente aperto il Navigation Drawer al fine di toccare la scheda chiamata "My Interests";

7. La pagina appena caricata ha lo stato [.MainActivity, ItemsOfInterestListFragment] e mostra una *Recycler View* in cui è presente l'oggetto a quale si ha espresso interesse allo step 2. Viene eseguito un tocco su tale oggetto;

8. La schermata aperta è la medesima descritta al punto 2. Si clicca nuovamente sul tasto "Indietro";

9. Tornati alla pagina precedente del punto 7, si apre ancora il Navigation Drawer, aprendo questa volta la scheda "Profile";

10. L'attuale pagina che ha come stato [.MainActivity, ShowProfileFragment] mostra i dati dell'utente corrente; in alto a destra si può trovare l'opzione di modifica, rappresentata tramite una matita: si procede toccando quest'opzione;

11. La nuova pagina si presenta con un layout simile a quella del punto 2, ma con alcune differenze nel tipo di dati da inserire: lo stato infatti ([.MainActivity, EditProfileFragment]) non è tra quelli finora visitati. Questa volta non vengono inseriti i dati, ma viene premuto il tasto "Indietro";

12. Di ritorno alla pagina di visualizzazione del profilo si termina la sessione di testing premendo sul tasto "Stop" presente nella toolbar di Scout.

L'albero di sessione risultante a questa sequenza di operazioni si può visualizzare graficamente in Figura 3.1. Si noti che le visite multiple di una stessa pagina non sono riportate all'interno dell'albero, poiché visitare nuovamente una pagina già aperta in precedenza non porta alla creazione di un nuovo ramo dell'albero.

È possibile ottenere un riepilogo delle pagine visitate durante la sessione anche nella forma di output testuale all'interno del terminale, invocando il metodo *printTree()* fornito dalla classe *Session*, che in questo caso specifico produrrebbe l'output presente in Figura 3.2. Tale output si ottiene effettuando una visita ricorsiva *pre-order* dell'albero ed evidenzia il livello di profondità di ciascuna pagina utilizzando un simbolo "—>" per ciascun livello superiore a quello di partenza.

```
"-1773264289"
--->"-1243657654"
--->"1726603052"
--->--->"-1111986024"
--->--->"-211126054"
--->--->--->"-1900209785"
--->--->--->--->"-1267504422"
```

Figura 3.2. Output della visita pre-order dell'albero della sessione.

Capitolo 4

Valutazione del plugin

Terminata la fase di teorizzazione e sviluppo del plugin è stata avviata la fase di sperimentazione su un campione di otto persone al fine di validare il lavoro svolto.

4.1 Selezione del Campione

Il tool all'interno del quale è stato implementato il plugin di Gamification è uno strumento professionale di GUI testing, quindi pensato per essere utilizzato da persone con un nutrito background di sviluppo software e di testing; per tale ragione i partecipanti all'esperimento sono stati scelti da un ambiente molto simile al target reale del tool. I soggetti sono stati per lo più studenti del corso di laurea magistrale in Ingegneria Informatica o in Informatica; una piccola parte del campione è invece composta da lavoratori nel mondo dell'informatica, con una laurea triennale o magistrale presso le facoltà sopra citate.

La partecipazione totale è stata di otto persone la cui età varia tra i 18 e i 30 anni; ciò influisce sull'esperienza e il background posseduti. Come si evince dai grafici mostrati in Figura 4.1, Figura 4.2 e Figura 4.3 (che mostrano rispettivamente le risposte alle domande 1.4, 1.5 e 1.8 del questionario in Tabella 4.1), tre dei partecipanti non hanno mai avuto esperienza di sviluppo di applicazioni, altri tre hanno avuto una breve esperienza della durata inferiore ad un anno, ma solo due persone hanno avuto a che fare con lo sviluppo di app per più di un anno. La maggior parte dei partecipanti ha sviluppato app utilizzando Java, solo tre persone su otto hanno utilizzato Kotlin o React Native come linguaggio di sviluppo, due delle quali affermano di aver sviluppato in entrambi i linguaggi. Si noti infine come

solo il 25% del campione affermi di aver avuto esperienza nel testing di app Android precedentemente.

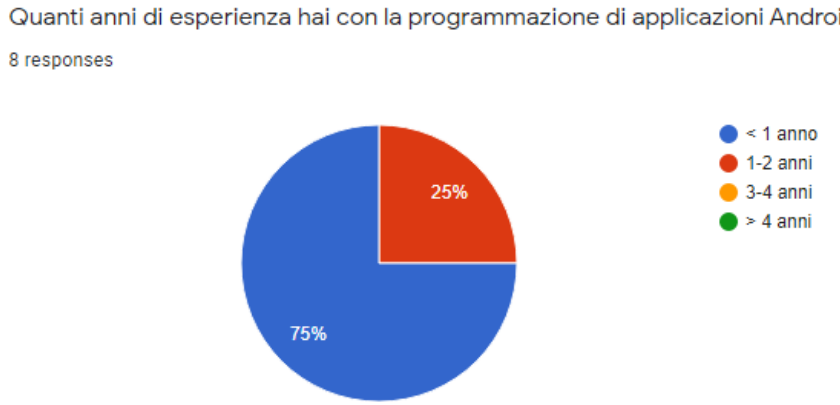

Figura 4.1. Anni di esperienza nello sviluppo di applicazioni Android

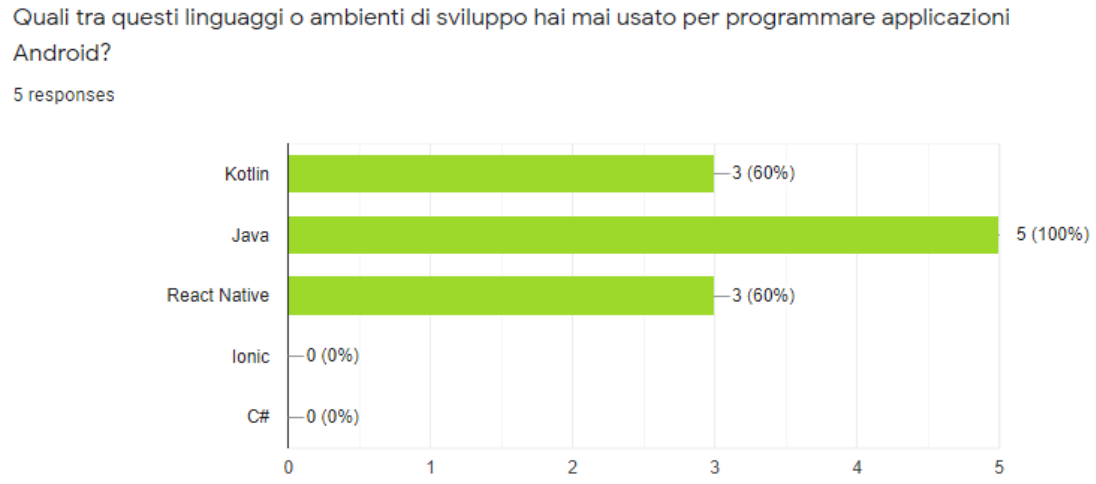

Figura 4.2. Diffusione dei linguaggi utilizzati per lo sviluppo di applicazioni Android

4.2 Metodologia

Per valutare il lavoro si è scelto uno schema di valutazione AB-BA in cui ogni partecipante avesse la possibilità di utilizzare il tool due volte: la prima volta su

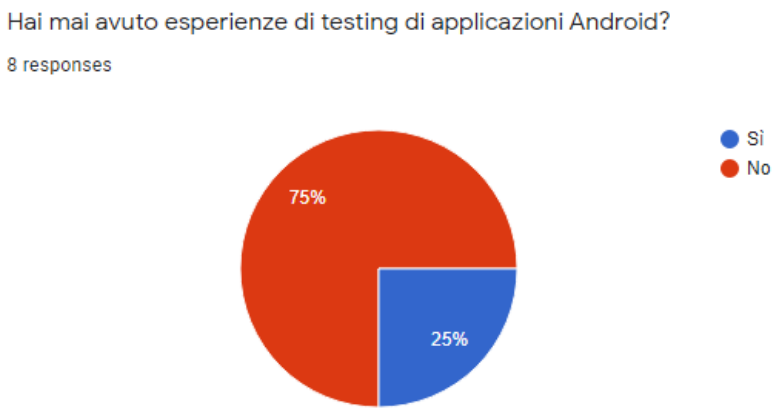

Figura 4.3. Percentuale di esperienza nel testing di applicazioni Android

un'applicazione A con la versione di Scout originale, la seconda volta su un'applicazione B utilizzando la versione ludicizzata di Scout. In questo modo il soggetto è in grado di comprendere la differenza tra l'utilizzo dello stesso strumento con e senza Gamification e di esprimere la propria opinione sull'impatto degli elementi ludici, sia in positivo che in negativo. Lo schema riassuntivo dell'esperimento è rappresentato in Figura 4.4.

A tutti i partecipanti è stato fatto un discorso introduttivo sull'utilizzo di Scout, in modo che fin dalla prima sessione fossero noti i meccanismi alla base dell'Augmented Testing e in particolare del funzionamento di Scout stesso. Successivamente, dopo aver verificato che l'ambiente di esecuzione fosse settato correttamente, si è lasciato libero il soggetto di effettuare una sessione di testing esplorativo della durata massima di 15 minuti all'interno di un'applicazione A.

Alla prima sessione è seguita una spiegazione riguardante gli elementi di Gamification presenti nel tool una volta attivato il plugin. Di essi sono stati resi noti sia l'aspetto che il funzionamento al fine di rendere consapevole il tester e di misurare in un secondo momento tramite il questionario (Tabella 4.1) l'efficacia del plugin percepita, invece dell'usabilità generale del software nella sua interezza.

La seconda sessione, infine, è stata eseguita dal tester con il plugin attivo su un'applicazione B, anch'essa della durata massima di 15 minuti. Si noti che le due sessioni hanno avuto SUT diversi per evitare che l'esperienza accumulata nella sessione sul primo applicativo potesse influenzare in qualche modo l'esperienza ludicizzata. Inoltre, il campione è stato diviso in due gruppi, il primo dei quali

ha eseguito il testing nell'ordine precedentemente descritto, il secondo invece nell'ordine inverso, testando prima l'applicazione B utilizzando Scout "as is" e poi l'applicazione A con il plugin di Gamification attivo.

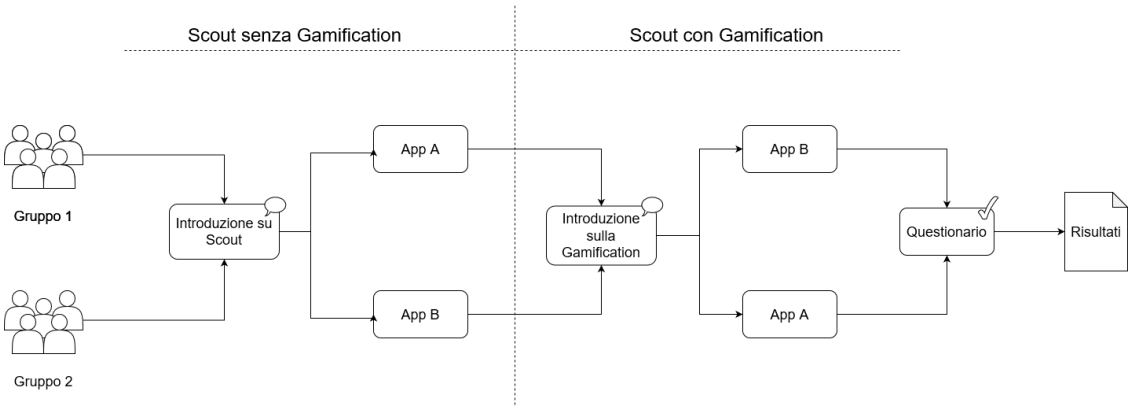

Figura 4.4. Organizzazione dell'esperimento.

Si noti che, nonostante gli elementi visivi di Gamification siano attivi solo nella seconda fase dell'esperimento, il tool è stato modificato in modo da produrre le metriche di interesse anche durante lo svolgimento della prima parte; ciò ha permesso un confronto sull'efficienza basato sui dati oggettivi, non solo sulla percezione dell'utente.

Le applicazioni utilizzate per le sperimentazioni sono **aMADzon2** (Cacciotto et al., 2020) e **Omninotes** (Federico Iosue, 2013), due app open source che differiscono per complessità in termini di Fragment utilizzati per rappresentare ogni pagina. La prima, infatti, utilizza una sola Activity, all'interno della quale un solo Fragment rimane in evidenza e cambia al cambiare della pagina; la seconda invece utilizza una Activity principale per le funzionalità primarie e una secondaria per gestire le impostazioni. All'interno dell'Activity principale vengono inoltre istanziati più Fragment per rappresentare la pagina. L'alternanza tra le applicazioni permette di valutare il plugin di Gamification senza il condizionamento derivante dall'architettura specifica dell'app.

La fase finale della sperimentazione è stata la compilazione del questionario mostrato in Tabella 4.1 tramite un Google Form in cui, dopo alcune domande introduttive sui dati anagrafici e sul background posseduto, è stata chiesta l'opinione del partecipante riguardo alla percezione dell'implementazione degli elementi ludici introdotti. È stato lasciato inoltre spazio alla segnalazione di eventuali problematiche

e imprevisti riscontrati, insieme ad uno spazio per consigli e suggerimenti.

Tabella 4.1: Domande del Questionario

ID	Domanda (Tipo)
1.1	Età (Scelta Multipla)
1.2	Hai mai avuto esperienze professionali con Java, come studente o lavoratore? (Scelta Multipla)
1.3	Quanti anni di esperienza hai con la programmazione in Java? (Scelta Multipla)
1.4	Quanti anni di esperienza hai con la programmazione di applicazioni Android? (Scelta Multipla)
1.5	Quali tra questi linguaggi o ambienti di sviluppo hai mai usato per programmare applicazioni Android? (Checkbox)
1.6	Hai mai avuto esperienze di testing di applicazioni Java? (Scelta Multipla)
1.7	Quali tool hai utilizzato per fare testing di applicazioni Java? (Aperta)
1.8	Hai mai avuto esperienze di testing di applicazioni Android? (Scelta Multipla)
1.9	Quali tool hai utilizzato per fare testing di applicazioni Android? (Aperta)
2.1	Hai compreso il modo di utilizzare Scout e il suo contesto di utilizzo? (Likert)
2.2	Hai trovato la Barra di Progresso utile a mostrare i tuoi progressi nel testing di una pagina? (Likert)
2.3	La Classifica finale ha stimolato il tuo senso di sfida verso altri tester passati? (Likert)
2.4	La Stella che contrassegna le pagine scoperte ti ha incoraggiato ad esplorare più approfonditamente l'applicazione da testare? (Likert)
2.5	Gli Easter Egg indotti ti hanno stimolato ad interagire con il maggior numero possibile di widget presenti all'interno dell'applicazione? (Likert)

Tabella 4.1: Domande del Questionario

ID	Domanda (Tipo)
2.6	Quali tra i seguenti elementi erano a te familiari prima dello svolgimento dell'esperimento? (Checkbox)
2.7	Quali, tra gli elementi selezionati nella risposta alla domanda precedente, sono stati utilizzati in maniera a te familiare? (Checkbox)
2.8	Quali tra i seguenti elementi sono per te essenziali in una sessione di testing? (Checkbox)
2.9	Hai trovato comprensibile la Schermata di Riepilogo e facilmente decodificabili le informazioni presenti al suo interno? (Likert)
2.10	Trovi che il tool utilizzato sia facilmente integrabile in un ambiente di testing esistente (lavorativo o di studio)? (Likert)
2.11	La versione di Scout che utilizza la Gamification ha aumentato la tua consapevolezza riguardo alla tua performance rispetto a quella senza? (Likert)
2.12	Nel complesso, trovi migliore la versione di Scout che utilizza la Gamification rispetto a quella senza? (Likert)
2.13	Hai riscontrato dei problemi durante lo svolgimento dell'esperimento? Se sì, descrivili brevemente (Aperta)
2.14	Hai dei suggerimenti per migliorare il tool proposto? (Aperta)

4.3 Ambiente di Esecuzione

Lo svolgimento dell'esperimento ha seguito la metodologia precedentemente definita. I partecipanti hanno svolto le sessioni di testing in un ambiente avente come principali caratteristiche quelle che seguono:

- CPU AMD Ryzen 5
- Sistema Operativo Windows 10 Home
- Dispositivo Android Emulato Pixel 2 API 29

- Android versione 10.0
- Appium Server versione 1.19.1

Essendo l'ambiente di esecuzione del plugin difficilmente riproducibile in maniera completamente integrata tramite virtualizzazione ed esigente di molte componenti da installare e configurare ad hoc, si è scelto di utilizzare sempre la medesima macchina per permettere ai soggetti coinvolti di effettuare le sessioni di prova senza dover perdere ulteriore tempo in configurazioni ed esecuzioni di script.

L'utilizzo di un unico PC ha permesso di salvare e mantenere lo stato interno del tool in maniera permanente; ciò fa sì che i risultati conseguiti dai primi tester vengano memorizzati e mostrati nella classifica in ogni momento successivo al completamento della sessione. Per non invalidare le sessioni svolte dai primi soggetti si è fatto in modo di partire da uno stato non nullo, ma da una classifica contenente quattro risultati (due per ogni app da testare) svolti nelle stesse tempistiche definite nella sezione precedente.

L'evoluzione dello stato ha fatto sì che i soggetti che provassero per ultimi il software fossero portati a competere con quelli precedenti; per verificare il senso di competizione sono state somministrate alcune domande che valutassero questo particolare aspetto.

I tester si sono quindi trovati ad eseguire i test sullo stesso ambiente di partenza, ma in condizioni di classifica e di esplorazione diverse.

4.4 Valutazione delle prestazioni ottenute

Per ogni tester sono stati raccolti i dati di entrambe le sessioni svolte, in particolare i dati della sessione non ludicizzata sono stati raccolti e trattati come se fosse effettivamente attiva la Gamification. Ciò ha permesso di valutare parametri aggiuntivi, oltre alla semplice coverage globale raggiunta, quali il punteggio attribuito alla sessione e il numero di easter egg totali scoperti.

L'elemento più significativo di confronto è senza dubbio la coverage, che misura in termini quantitativi la percentuale di esplorazione raggiunta dal tester durante la sessione. Nelle due sessioni a confronto tutti gli individui tranne uno hanno avuto un miglioramento nella coverage raggiunta, toccando addirittura il doppio in alcuni casi come si evince in Figura 4.5.

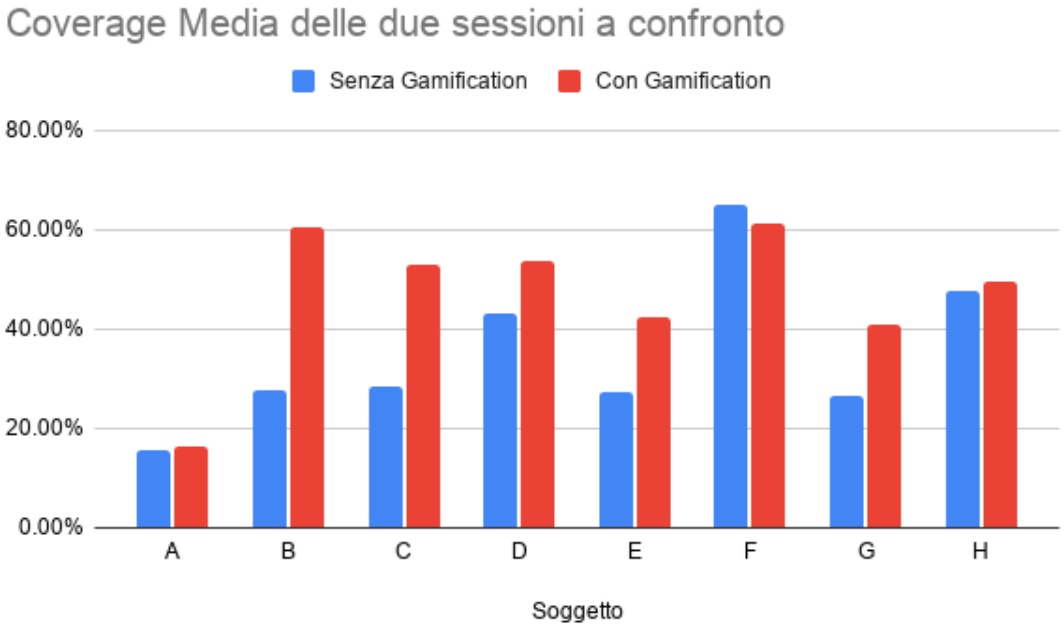

Figura 4.5. Coverage a confronto con e senza plugin di Gamification

Se si analizza l'incremento percentuale mostrato nella Figura 4.6 subìto dalla coverage si nota come la metà dei soggetti abbia avuto un incremento di oltre il 50% nella coverage raggiunta, con un picco per l'individuo B che ha totalizzato oltre il doppio con il plugin attivo. La restante metà è composta da un soggetto che ha subìto un lieve peggioramento, due che hanno avuto un miglioramento minimo, quasi impercettibile, e un soggetto che ha migliorato del 25% il proprio risultato. Questi dati mostrano come la coverage abbia una tendenza a migliorare grazie agli elementi ludici introdotti: infatti l'incremento medio riferito alla coverage ottenuto dopo l'applicazione della Gamification è del 43%.

Nonostante la coverage sia di per sé un buon indicatore della qualità della sessione di testing, essendo ottenuta tramite una media tra le varie pagine visitate, non si ha un indicatore puntuale della coverage raggiunta nelle singole pagine.

È opportuno confrontare, oltre alla coverage media, il punteggio calcolato mediante la Formula (2.1). Questo parametro fornisce informazioni qualitative supplementari a quelle fornite dalla coverage media in quanto integra anche la componente esplorativa data da widget e pagine scoperte (Formula (2.4)) e la componente di efficienza (Formula (2.5)).

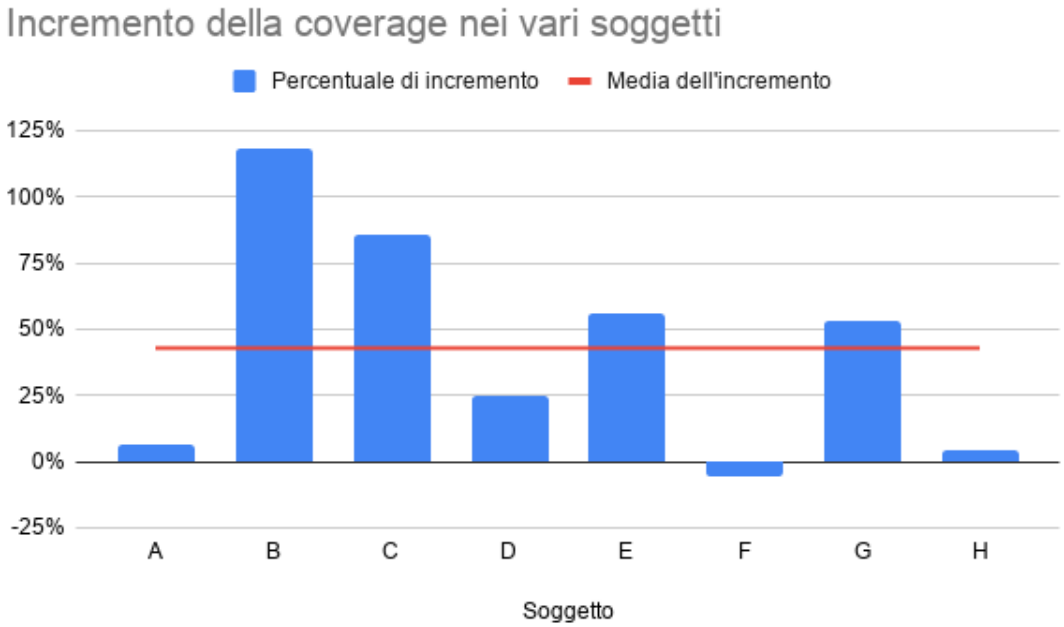

Figura 4.6. Incremento nella coverage nei vari soggetti e incremento medio

Come si evince dalla Figura 4.7, il punteggio dei tester in presenza degli elementi ludici è generalmente aumentato, fatta eccezione per un individuo che ha subìto un lieve peggioramento di un solo punto. Il raggiungimento di punteggi migliori, di pari passo con l'aumento della coverage, mostra come la sessione di testing abbia beneficiato degli elementi ludici, portando il tester al raggiungimento di performance superiori. Nel confronto va tenuto presente che i due punteggi ottenuti riguardino applicazioni differenti che, seppur presentino caratteristiche simili, sono state sviluppate seguendo diversi approcci. Tale disparità presente nei soggetti di testing può aver influito nella variazione delle performance, sia in positivo che in negativo.

Un ultimo dato che può essere analizzato è il numero di easter egg attivati durante la sessione. Questo numero assoluto mostra quanto sia stata profonda l'esplorazione dei widget di una pagina: infatti in ogni pagina è nascosto un easter egg attivabile cliccando su un widget scelto casualmente. Ne consegue che maggiore è l'insieme di widget cliccati, maggiore sarà la probabilità di attivare l'elemento nascosto.

L'attivazione degli easter egg nelle varie pagine, come citato in precedenza, è

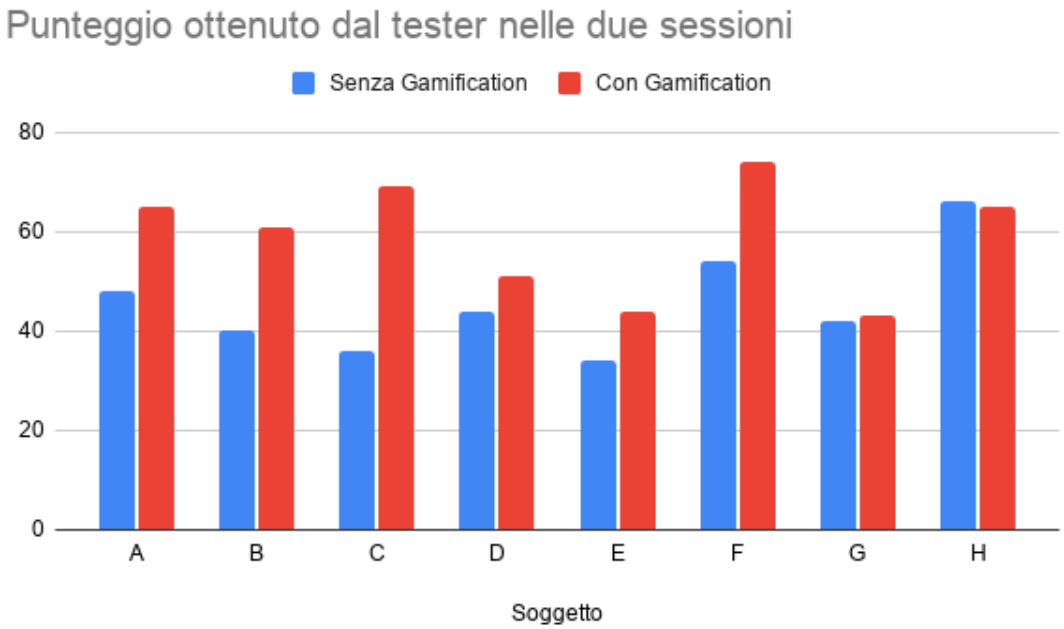

Figura 4.7. Punteggio a confronto con e senza plugin di Gamification

avvenuta anche a Gamification disabilitata: infatti la disattivazione del plugin ha semplicemente reso invisibili gli elementi grafici, permettendo il confronto tra i dati raccolti in entrambe le occasioni.

Come possiamo notare in Figura 4.8, quando il tester è stato messo al corrente della presenza di easter egg nascosti nell'applicazione da testare, la ricerca dell'elemento ha stimolato un numero di interazioni maggiore nella pagina, portando non solo all'individuazione dell'elemento ricercato, ma anche ad una coverage maggiore all'interno delle singole pagine.

Confrontando il numero totale di widget con i quali il tester ha interagito nelle due sessioni di testing (Figura 4.9), si può notare come la quasi totalità dei soggetti abbia avuto un incremento significativo nelle interazioni avvenute.

Si noti che non è stato presentato nessun confronto per quanto riguarda i dati di esplorazione delle pagine totali poiché nessun miglioramento significativo è stato riscontrato nelle applicazioni testate. Ciò si può facilmente spiegare notando come le applicazioni target **aMADzon2** e **Omninotes** siano costituite da un esiguo insieme di pagine (in base alla definizione che le lega allo stato dell'applicazione, proposta nella sezione 2.4).

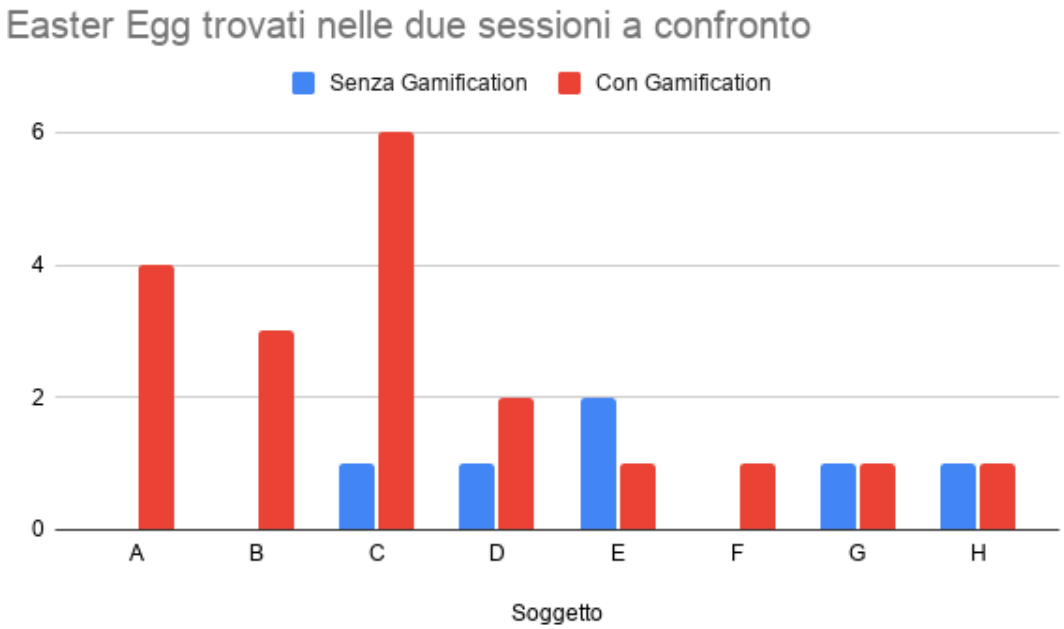

Figura 4.8. Numero di Easter Egg a confronto con e senza plugin di Gamification

Si può concludere che buona parte ha migliorato la propria prestazione a seguito dell'introduzione della Gamification dal punto di vista di almeno uno dei parametri considerati.

4.5 Valutazione degli Elementi di Gamification

Uno dei motivi principali che ha portato alla diffusione dell'approccio ludico è la stimolazione della componente emotiva suscitata nell'utente, che si manifesta a seguito dell'interazione con gli elementi di Gamification. Questi elementi riescono a dare risultati importanti se ben progettati e implementati, ma rischiano di risultare un peso se il loro impiego fosse mal gestito, come si è mostrato nella sottosezione 1.2.5. Per valutare la percezione dell'efficacia dell'implementazione proposta per ogni elemento ludico, sono state somministrate le domande dalla 2.2 alla 2.5 della Tabella 4.1.

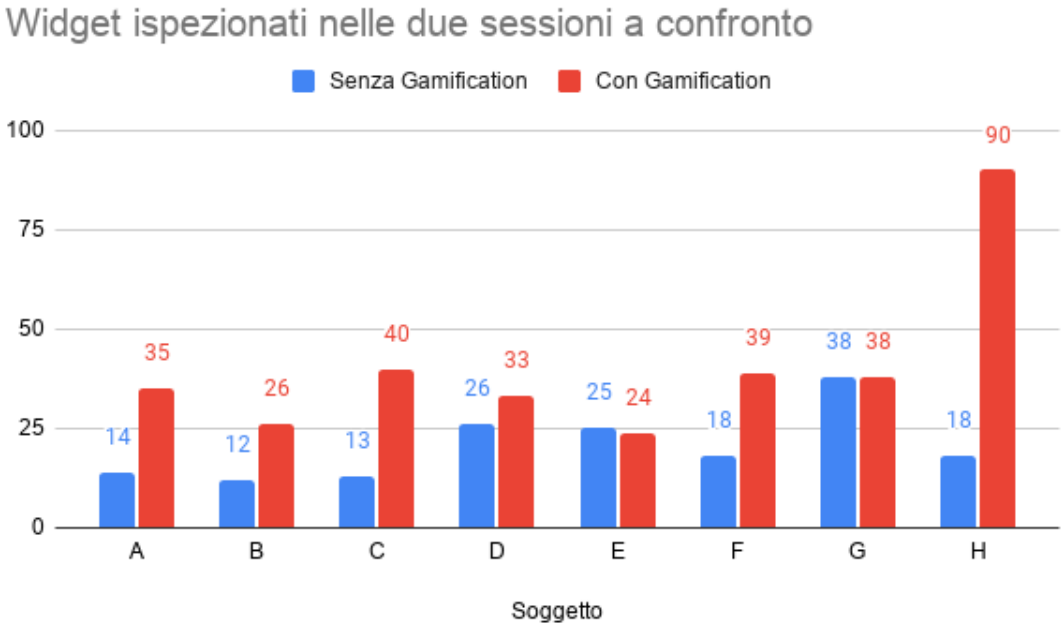

Figura 4.9. Confronto tra il numero widget che hanno subìto interazioni con e senza plugin di Gamification

4.5.1 Barra di Progresso

La barra di progresso è stata considerata all'unanimità positiva per il contributo che ha dato nel mostrare la coverage raggiunta in tempo reale sulla pagina corrente: la metà del campione ha dato il voto massimo, mentre l'altra metà si è espressa comunque in maniera favorevole come si evince in Figura 4.10.

Tra i commenti ricevuti nello spazio dedicato ai suggerimenti, tre persone hanno suggerito piccole modifiche grafiche che rendessero maggiormente visibile la barra di completamento. Le proposte riguardano un aumento dello spessore ed un meccanismo di blocco della barra, in modo tale che, anche a seguito di azioni di scroll della finestra, la barra rimanga ancorata nello stesso punto dell'interfaccia. Tutti e tre i suggerimenti provengono da persone che hanno espresso il voto di approvazione più blando.

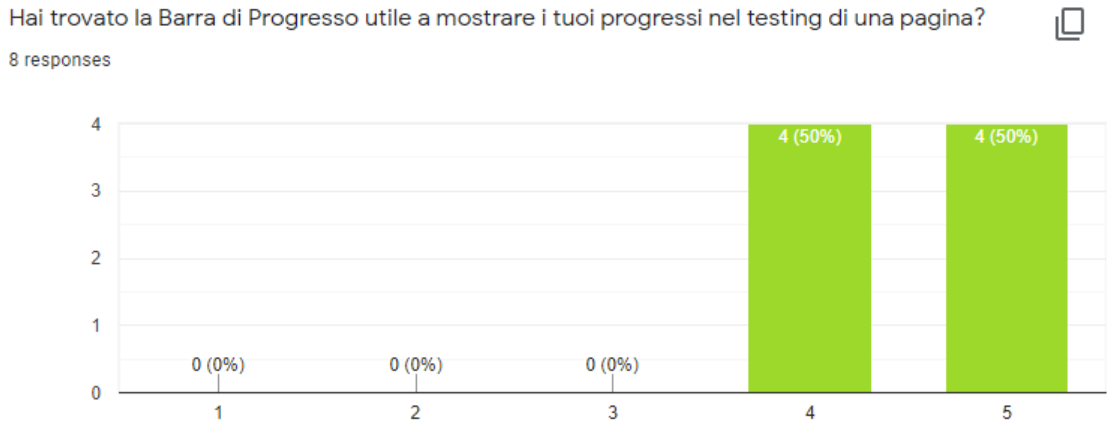

Figura 4.10. Valutazione Likert della Barra di Progresso

4.5.2 Classifica

La classifica è stato l'elemento che più di tutti ha ricevuto la massima approvazione dai soggetti coinvolti (Figura 4.11), il 75% del campione, mentre un solo individuo ha segnalato che la presenza di una classifica non abbia stimolato un vero senso di sfida, ma più un banale senso di curiosità nello scoprire la posizione occupata. Al contrario, invece, due soggetti hanno affermato che la presenza di una classifica li spingerebbe a effettuare nuove sessioni di testing in modo più approfondito per migliorare il proprio punteggio.

Un suggerimento piuttosto interessante viene riportato di seguito: "*sarebbe interessante scomporre la classifica secondo i diversi parametri mostrati nella schermata di riepilogo e attribuire il voto anche ad essi, così da poter confrontare più aspetti e non solo il punteggio totale*".

La proposta dovrebbe essere facilmente attuabile, essendo le metriche di interesse già calcolate e implementate. Ulteriore lavoro è richiesto nel rendere confrontabili in modo assoluto e indipendente dal SUT alcune metriche (le pagine totali visitate, le pagine nuove scoperte, i widget scoperti).

4.5.3 Easter Egg

L'iniezione di Easter Egg visivi nell'applicazione è stata recepita positivamente da tre quarti del campione (Figura 4.12), la metà dei quali lo approva totalmente, l'altra metà esprime un'approvazione più lieve. I due tester restanti che hanno

Figura 4.11. Valutazione Likert della Classifica

espresso un giudizio leggermente negativo, uno ha esposto dei dubbi riguardo al meccanismo di generazione degli easter egg, basato sull'estrazione a sorte tra i widget cliccabili presenti in pagina. Tale perplessità potrebbe essere dovuta alla mancanza di familiarità con tale elemento ludico, come egli stesso ha dichiarato alle domande 2.6 e 2.7 della Tabella 4.1.

L'altro soggetto che segnala di non essere stato spinto realmente all'esplorazione dalla presenza di easter egg motiva la sua risposta dichiarando quanto segue: "*Gli Easter Egg non mi hanno stimolato ad esplorare l'App quanto la Barra di Progresso*".

4.5.4 Stella

L'utilizzo di un simbolo a forma di stella per marcare le pagine nuove è l'elemento che maggiormente ha diviso l'opinione del campione. Il 25% di esso, infatti, ha espresso una totale disapprovazione, il 50% si è mostrata favorevole in modo contenuto e la restante parte è divisa tra una totale approvazione e l'astensione come mostrato in Figura 4.13.

Il motivo di tale eterogeneità di risposte va ricercato nel meccanismo di funzionamento della stella e nella maniera in cui è stato svolto l'esperimento: ciò ha influenzato in maniera determinante la sessione di testing ludicizzata. La presenza di pagine nuove dipende infatti dallo stato del plugin di Gamification che è dato dall'unione degli stati dei tester precedenti rispetto al tester corrente. Per tale

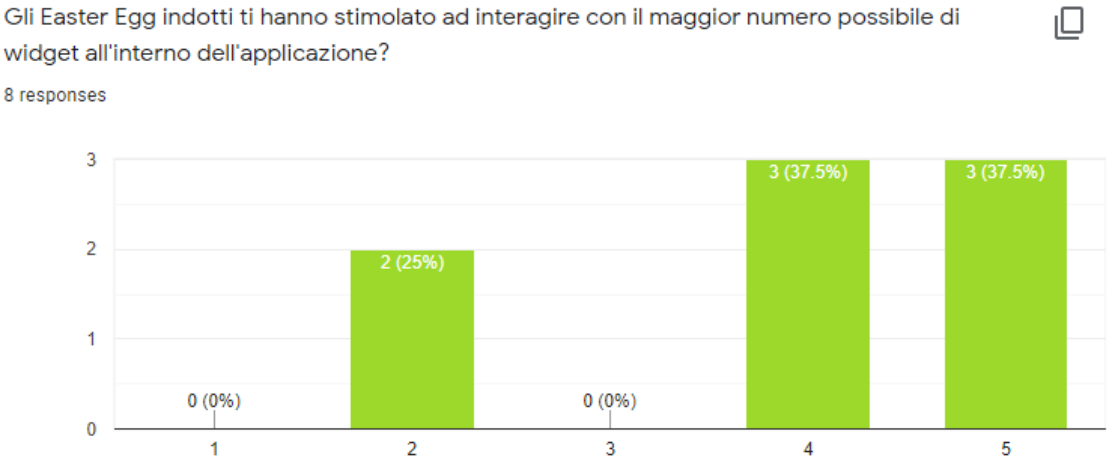

Figura 4.12. Valutazione Likert dell'Easter Egg

motivo, la probabilità di scoprire pagine nuove è più alta per i tester che effettuano per primi le sessioni di testing, mentre è minore per gli ultimi tester in ordine cronologico.

Gli individui che hanno espresso un parere negativo hanno infatti motivato la loro risposta affermando di non essere riusciti a trovare pagine contrassegnate come nuove durante la sessione svolta; al contrario, il riscontro è stato positivo per i soggetti che hanno effettivamente trovato pagine mai esplorate.

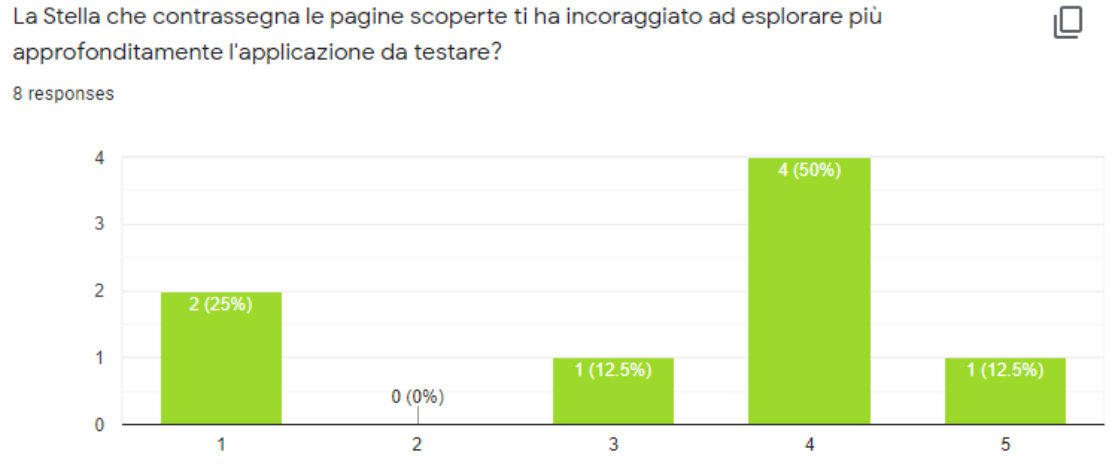

Figura 4.13. Valutazione Likert della Stella

4.6 Valutazione complessiva del tool

Una valutazione globale di alcuni aspetti del tool per come è stato percepito dai tester è necessaria per stabilire se le varie meccaniche ludiche siano state integrate in modo efficace nel software esistente.

Come si può notare in Figura 4.14, il campione concorda all'unanimità sul fatto che il plugin abbia aumentato la consapevolezza della performance in atto. Dai commenti liberi si apprende inoltre che la consapevolezza derivi principalmente dalla barra di progresso e dalla stella che contrassegna le pagine nuove.

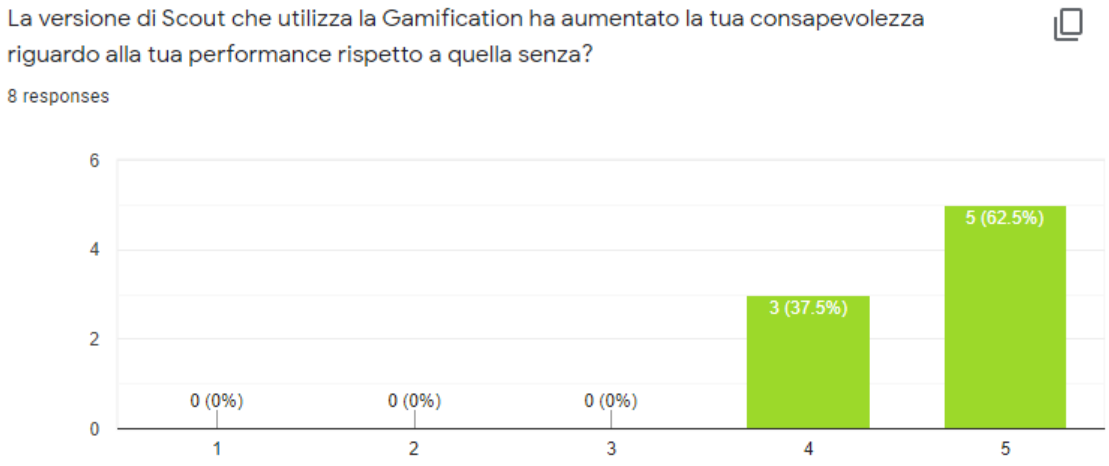

Figura 4.14. Valutazione Likert della consapevolezza trasmessa

Analizzando le risposte ricevute alla domanda 2.8 del questionario in Tabella 4.1 (mostrate in Figura 4.15) si nota come l'intero campione sia concorde sull'essenzialità della barra di progresso. Ciò mostra l'utilità di un meccanismo di segnalazione di progressi in tempo reale che permetta di percepire quanto, in un determinato momento, sia esaustivo l'operato del tester su una pagina. La presenza di un punteggio e di una classifica basata su di esso è ritenuta essenziale da più della metà dei partecipanti. Questo mette in luce quanto lo spirito di competizione e di confronto sia importante per i tester, che al termine della sessione, appena appresa la propria posizione in classifica, hanno espresso approvazione o lamentele nei confronti del loro operato, mostrandosi desiderosi di occupare le posizioni più alte della classifica finale.

Minor consenso è stato ricevuto invece dal meccanismo di identificazione delle pagine nuove e dalla caccia agli easter egg, che vengono considerati essenziali solo dal 25% del campione totale. Si possono considerare pertanto tali elementi come un surplus che non costituisce lo scheletro del plugin, ma che arricchisce l'esperienza dell'utente. La ristrettezza dell'insieme delle pagine individuate dal proprio stato e la semplicità grafica dell'easter egg, possono aver influito sulla percezione di tali elementi come superflui rispetto agli altri. Ciò potrebbe lasciare un margine di miglioramento a tali meccaniche che, adeguatamente sviluppate, potrebbero incontrare un consenso più ampio.

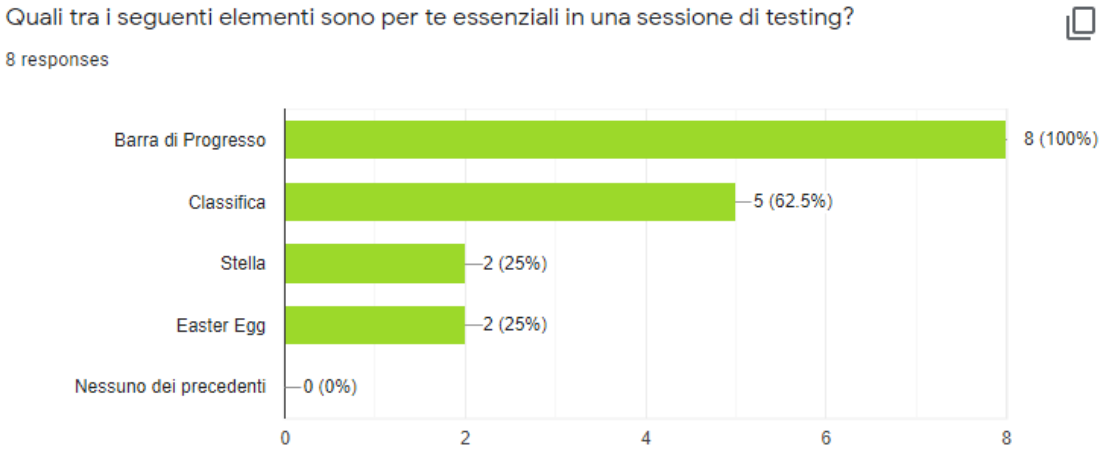

Figura 4.15. Essenzialità dei singoli elementi ludici

Un ulteriore aspetto che merita di essere analizzato è l'integrabilità del tool proposto in un ambiente di sviluppo noto: infatti, nonostante il miglioramento delle performance sia evidente, l'utilizzo del tool proposto, se non ben integrato in un ambiente esistente, può non trovare applicazione in contesti reali di utilizzo quali quello accademico e industriale.

Come mostrato nella Figura 4.16, il campione si divide equamente in due sul tema dell'integrabilità: metà, infatti, ritiene che l'adozione sia abbastanza facile, l'altra invece metà ritiene complesso l'inserimento, la configurazione e l'utilizzo del tool per come è stato mostrato nel prototipo.

I pareri negativi non sorprendono in quanto, fin da principio, è noto che la procedura di comunicazione tra l'ambiente Android simulato e Scout sia innegabilmente complessa nella configurazione; inoltre, l'emulazione dell'app attraverso

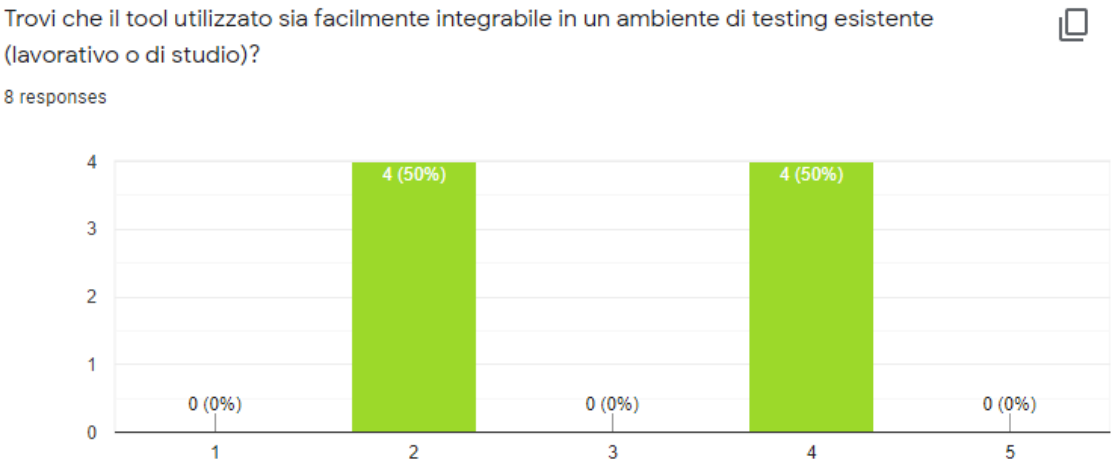

Figura 4.16. Valutazione Likert dell'integrabilità del tool

Scout costituisce un'ulteriore fonte di rallentamento nelle performance del tool a causa del meccanismo basato su continui screenshot dell'applicazione originale.

È d'altra parte confortante l'approvazione ricevuta dalla metà del campione che ritiene la complessità totale del software ragionevole al punto da poter comunque essere utilizzato in un contesto noto.

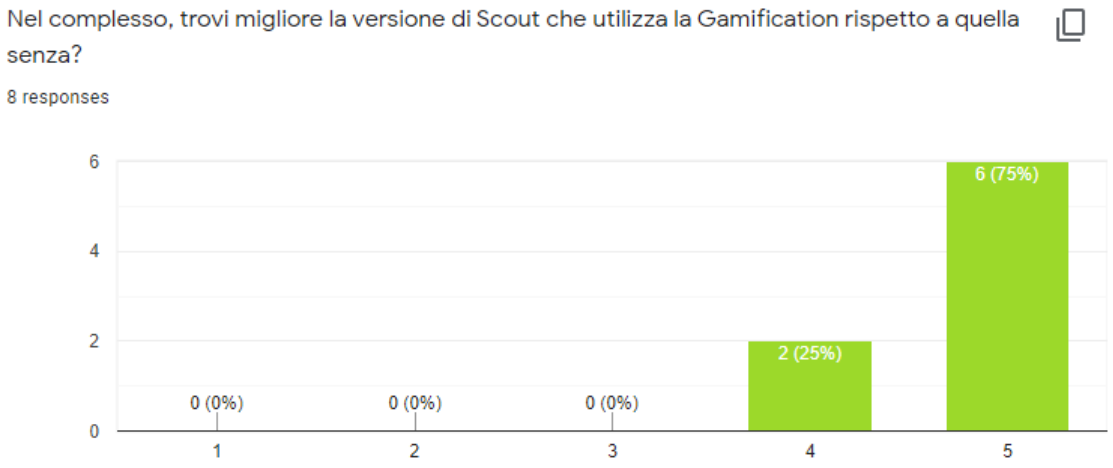

Figura 4.17. Preferenza del tool con e senza plugin di Gamification

Infine, alla domanda 2.12 della Tabella 4.1, si nota l'unanime percezione positiva del plugin sviluppato (Figura 4.17): infatti, confrontando l'esperienza del tester

prima e dopo l'introduzione degli elementi ludici, tutti i partecipanti ammettono di trovare migliore il tool nel complesso quando la Gamification è stata attivata rispetto al tool originale.

Capitolo 5

Conclusioni

Alla luce degli incoraggianti risultati ottenuti vanno sottolineate le limitazioni e le fragilità riscontrate, al fine di gettare le basi per lavori futuri di potenziamento e di espansione del lavoro presentato.

5.1 Limiti di applicabilità interna

La principale fonte di fragilità e di spunti per lavori futuri è data dalla componente di integrazione tra Scout e l'ambiente di esecuzione dell'app. L'utilizzo di Appium permette infatti di analizzare il codice XML associato alla schermata mostrata in un determinato momento nell'AVD, ma non fornisce nessun meccanismo automatizzato di ricostruzione dell'ambiente grafico dell'app. La soluzione che è attualmente implementata si basa sull'acquisizione di uno screenshot ad ogni ciclo di refresh, il salvataggio di tale immagine in un file e la sua successiva lettura e proiezione nella schermata di Scout. Questo meccanismo, svolto dentro all'*AppiumPlugin*, introduce una latenza non banale, quantificata sperimentalmente durante lo sviluppo nell'ordine dei 500 millisecondi.

Una soluzione sarebbe la ricerca di un meccanismo capace di incorporare l'operazione di acquisizione dell'immagine presente sullo schermo con Appium, in modo da ottimizzare le operazioni esistenti.

Un altro problema legato al ritardo introdotto in Scout deriva dalla mancanza, in Appium, di strumenti di rilevamento dei Fragment nell'Activity corrente. La soluzione attualmente implementata analizza l'output di un comando lanciato nella shell di sistema facendo un parsing, che restituisce la rappresentazione dei Fragment

e delle Activity in esecuzione sotto forma di insieme di stringhe. Durante la fase di sviluppo di questo modulo parser, si è notato come l'output ricevuto in seguito all'esecuzione del comando di shell di tanto in tanto contenesse delle segnalazioni di errori transitori. Il meccanismo di recovery da questa situazione consiste nell'invocazione ricorsiva della stessa funzione, nella speranza che il comando, lanciato nuovamente, restituisca un output non affetto da errori. Si è notato come l'errore transitorio si risolva col passare del tempo: sperimentalmente in un massimo di quattro iterazioni della funzione ricorsiva l'errore transitorio si estingue, fornendo l'output corretto.

La latenza misurata sperimentalmente derivante dal meccanismo di riconoscimento dei Fragment attivi è quantificata nell'ordine dei 100-150 millisecondi per ogni iterazione del parsing. Essendo questo meccanismo di riconoscimento dei Fragment e delle Activity necessario ad ogni click per rilevare se la pagina sia cambiata rispetto a quella precedente, non è trascurabile il ritardo accumulato con il parsing, soprattutto se le iterazioni necessarie ad ottenere l'output corretto dal comando di shell sono multiple.

Tale ritardo, se sommato a quello precedentemente descritto, lede in modo non banale l'esperienza del tester: infatti, nel questionario sottoposto al campione, più di metà dei partecipanti ha segnalato tale problema di performance, lamentando che in un contesto di utilizzo reale non sia accettabile una reattività così bassa. Un sistema di riconoscimento dell'Activity e dei Fragment attivi permetterebbe di evitare il parsing dell'output del comando di shell, attualmente implementato con un algoritmo piuttosto fragile.

Va riconosciuto inoltre come la rappresentazione della pagina associata univocamente allo stato, definito come insieme di Fragment attivi nell'Activity in esecuzione, si sia dimostrata non del tutto adeguata in quanto sono state notate e segnalate sperimentalmente dal campione diverse occasioni in cui, al mutare dell'interfaccia del SUT (soprattutto all'apertura dei menù), l'interfaccia di Scout non si adattasse di conseguenza, mostrando in evidenza widget appartenenti a pagine visitate in precedenza. Questa mancanza di sincronia può pregiudicare il corretto riconoscimento dei widget presenti nella pagina e di conseguenza il corretto funzionamento del tool.

Nelle app Android, inoltre, i Fragment sono spesso riutilizzati per rappresentare concetti diversi. Il riciclo di tali strumenti porta delle ottimizzazioni per il

programmatore che deve scrivere il codice, ma complica il meccanismo di discernimento di una pagina nuova da una già testata. Un meccanismo di riconoscimento della pagina che non sia puramente basato sul proprio stato, ma che la leghi al proprio contenuto, permetterebbe di riconoscere pagine ritenute uguali a causa della corrente rappresentazione.

È opportuno trovare una rappresentazione alternativa della pagina dell'app da testare o integrare quella attualmente in uso con informazioni supplementari, in modo tale che le pagine rappresentate nello stato di Scout corrispondano effettivamente a quelle reali dell'applicazione, includendo tutti i tipi di menù.

Il plugin presentato, per come è stato concepito, non ammette un utilizzo simultaneo da diversi dispositivi, nonostante questo sia un tipico scenario di utilizzo. L'abbandono della memorizzazione su file, in favore dell'adozione di un database remoto apre le porte alla condivisione delle statistiche e dei punteggi. Tuttavia, un sistema di sincronizzazione tra le sessioni concorrenti di testing sulla stessa applicazione target deve essere sviluppato per poter gestire le metriche che dipendono dall'istante di tempo in cui avviene l'interazione (scoperta di pagine e widget nuovi).

5.2 Limiti di applicabilità esterna

L'utilizzo dell'ambiente Android emulato insieme a Scout richiede un ambiente di esecuzione piuttosto performante: si è notato infatti come l'esecuzione su un PC avente un tradizionale HDD sia oltremodo rallentata, al punto da essere considerata inutilizzabile. è perciò necessaria la presenza di un SSD per supportare l'AVD.

Un'alternativa valida non ancora prevista è l'utilizzo di un dispositivo fisico in modalità debug collegato tramite il cavo USB. Tale opzione libera il PC utilizzato dall'onere dell'archiviazione ed emulazione di tutto l'ambiente Android, permettendo inoltre di implementare il principio di **Separation of Concerns**, lasciando agli sviluppatori solamente la responsabilità di installare la versione dell'app da testare nel dispositivo fisico, evitando che il tester debba attingere al codice sorgente da compilare o al file APK di installazione.

La corrente emulazione di Android che il tool effettua è monca di azioni basate su gesti più complessi del semplice tocco: gesti come lo swipe, il pinch e il tocco prolungato non sono infatti ancora riconosciuti e implementati. L'emulazione e il riconoscimento di queste azioni sono necessari in un contesto di utilizzo reale, poiché sono sfruttate in maniera preponderante dalle app moderne per funzionalità di

base, come ad esempio lo scroll della schermata, ma anche come shortcut di operazioni che altrimenti richiederebbero molti più passaggi. Assicurarsi del corretto funzionamento di questi simboli è parte del lavoro del tester, che non deve mai trascurare nessuna delle possibili interazioni previste.

Va infine segnalato come i risultati incoraggianti mostrati nella sezione 4.5 siano comunque frutto di una sperimentazione limitata, concentratasi su un piccolo campione e su due sole App. Per avere una conferma del miglioramento delle prestazioni derivanti dall'applicazione della Gamification alla disciplina del GUI testing è necessaria una sperimentazione più ampia di quella presentata. L'impiego di un numero maggiore e più eterogeneo di App target per confermare o smentire i risultati ottenuti in termini di prestazioni ottenute è lasciato a sviluppi futuri.

5.3 Lavori Futuri

Il lavoro presentato fornisce un incipit di applicazione della Gamification al GUI testing. Alla luce dei risultati positivi riscontrati a seguito della sperimentazione, vengono di seguito elencati i temi su cui dovrebbero concentrarsi i lavori futuri, al fine di smussare i limiti precedentemente evidenziati:

- Teorizzazione di una rappresentazione più efficace delle pagine del SUT, in modo tale da includere anche i menù nel modello.

- Ricerca di un metodo più rigoroso e deterministico per l'ottenimento dello stato dell'app.

- Miglioramento dell'aspetto grafico di easter egg, stella e classifica. Quest'ultima dovrà inoltre essere suddivisa per app e comprenderà la possibilità di sviscerare il punteggio nelle sottocomponenti e visualizzare il voto per ognuna di esse.

- Potenziamento dell'emulazione, includendo nuovi gesti a supporto delle piene funzionalità fornite dal SUT.

- Ricerca ed eventuale implementazione di un sistema ottimizzato di acquisizione dell'interfaccia originale del SUT all'interno di Scout.

- Implementazione della memorizzazione dei dati tramite un database centralizzato, con conseguente gestione della sincronizzazione tra le sessioni di testing effettuate sulla stessa app target.
- Organizzazione di una sperimentazione più ampia e approfondita che coinvolga un numero maggiore di soggetti e un insieme di SUT più ampio per poter confermare i risultati ottenuti.

Riferimenti bibliografici

Anderson P., N. T., & R., M. (2015, 06). Facilitating programming success in data science courses through gamified scaffolding and learn2mine. In (p. 99-104). doi: 10.1145/2729094.2742597

Berkling, K., & Thomas, C. (2013). Gamification of a software engineering course and a detailed analysis of the factors that lead to its failure. In *2013 international conference on interactive collaborative learning (icl)* (p. 525-530).

Borjesson, E., & Feldt, R. (2012). Automated system testing using visual gui testing tools: A comparative study in industry. In *2012 ieee fifth international conference on software testing, verification and validation* (p. 350-359). doi: 10.1109/ICST.2012.115

Buckley, I., & Clarke, P. (2018, 01). An approach to teaching software testing supported by two different online content delivery methods.. doi: 10.18687/LACCEI2018.1.1.377

Cacciotto, F., Fulcini, T., Piacentini, G., & Pietrasanta, M. (2020). *amadzon2.* `https://bitbucket.org/PoliAL-MAD2020/project_03/`. (An App for buying and selling items, developed for academic purpose)

Costa, I., & Oliveira, S. (2019). A systematic strategy to teaching of exploratory testing using gamification. In *Proceedings of the 14th international conference on evaluation of novel approaches to software engineering* (p. 307–314). SCITEPRESS - Science and Technology Publications, Lda. doi: 10.5220/0007711603070314

Dal Sasso, T., Mocci, A., Lanza, M., & Mastrodicasa, E. (2017). How to gamify software engineering. In *Saner 2017 - 24th ieee international conference on software analysis, evolution, and reengineering* (p. 261-271). (Cited By :15) doi: 10.1109/SANER.2017.7884627

Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011, 09). From game design elements to gamefulness: Defining gamification. In (Vol. 11, p. 9-15). doi: 10.1145/2181037.2181040

Federico Iosue, O. (2013). *Omninotes.* `https://github.com/federicoiosue/Omni-Notes`. (An open source App with a simple interface for notes and

todos)

Fowler, M. (1999). *Refactoring: Improving the design of existing code.* Addison-Wesley Longman Publishing Co., Inc.

Fraser, G., Gambi, A., Kreis, M., & Rojas, J. M. (2019). Gamifying a software testing course with code defenders. In *Proc. of the acm technical symposium on computer science education (sigcse).* ACM. (To appear)

Herranz, E., Colomo-Palacios, R., & Amescua-Seco, A. (2013). Towards a new approach to supporting top managers in spi organizational change management. *Procedia Technology*, *9*, 129 - 138. doi: https://doi.org/10.1016/j.protcy.2013.12.014

Herranz, E., Colomo-Palacios, R., de Amescua, A., & Yilmaz, M. (2014, 06). Gamification as a disruptive factor in software process improvement initiatives. *Journal of Universal Computer Science*, *20*, 885-906.

Mollick, E., & Rothbard, N. (2014, 05). Mandatory fun: Consent, gamification and the impact of games at work.
doi: 10.2139/ssrn.2277103

Nass, M., Alégroth, E., & Feldt, R. (2019). Augmented testing: Industry feedback to shape a new testing technology. In *2019 ieee international conference on software testing, verification and validation workshops (icstw)* (p. 176-183). doi: 10.1109/ICSTW.2019.00048

Nass, M., Alégroth, E., & Feldt, R. (2020). On the industrial applicability of augmented testing: An empirical study. In *2020 ieee international conference on software testing,verification and validation workshops (icstw)* (p. 364-371). doi: 10.1109/ICSTW50294.2020.00065

Passos E.B., N. P., Medeiros D.B., & E.W.G., C. (2011). Turning real-world software development into a game. In *2011 brazilian symposium on games and digital entertainment* (pp. 260–269).

Pedreira, O., García, F., Brisaboa, N., & Piattini, M. (2015). Gamification in software engineering – a systematic mapping. *Information and Software Technology*, *57*, 157 - 168. doi: 10.1016/j.infsof.2014.08.007

Platonova, V., & Bērziša, S. (2017, 12). Gamification in software development projects. *Information Technology and Management Science*, *20*. doi: 10.1515/itms-2017-0010

Prasetya, W., Leek, C., Melkonian, O., ten Tusscher, J., van Bergen, J., Everink, J., ... van Zon, W. (2019). Having fun in learning formal specifications. In *2019 ieee/acm 41st international conference on software engineering: Software engineering education and training (icse-seet)* (p. 192-196). doi: 10.1109/ICSE-SEET.2019.00028

Rojas, J. M., & Fraser, G. (2016). Code defenders: A mutation testing game. In *Proc. of the 11th international workshop on mutation analysis* (pp. 162–167). IEEE. doi: 10.1109/ICSTW.2016.43.

Santos, H., Durelli, V., Souza, M., Figueiredo, E., Silva, L., & Durelli, R. (2019, 09). Cleangame: Gamifying the identification of code smells. In (p. 437-446). doi: 10.1145/3350768.3352490

Sheth, S., Bell, J., & Kaiser, G. (2011). Halo (highly addictive, socially optimized) software engineering. In (p. 29–32). Association for Computing Machinery. doi: 10.1145/1984674.1984685

Verma, N. (2017). *Mobile test automation with appium.* Packt.

Zsigmond, I., Bocicor, M., & Molnar, A.-J. (2020, 01). Gamification based learning environment for computer science students. In (p. 556-563). doi: 10.5220/0009579305560563