



UNIVERSITÀ DEGLI STUDI DI BERGAMO

Scuola di Ingegneria

Corso di laurea in Ingegneria Informatica

Classe N. L-08 - Ingegneria dell'informazione (D.M. 270/04)

Tesi di laurea triennale

3DSAT

Progettazione e sviluppo di 3D Stereo Acuity Test con algoritmo
PEST (Parameter Estimation by Sequential Testing)

Relatori

prof. Angelo GARGANTINI

Correlatore:

prof.ssa Silvia BONFANTI

Laureandi

Samuele FERRI [1045975]

Simone SUDATI [1045936]

ANNO ACCADEMICO 2018-2019

Indice

1	Introduzione generale	1
1.1	Ambliopia	1
1.2	Stereoacuità	3
1.3	Anaglifi	5
1.4	Il progetto 3D4AMB	6
1.5	Il nostro apporto	7
2	Applicazione	9
2.1	Introduzione all'applicazione	9
2.2	Main activities	10
2.2.1	Login	10
2.2.2	Main Doctor	11
2.2.3	Main Patient	12
2.2.4	Test	13
2.2.5	Results	15
2.2.6	Settings	16
2.3	Util activities	17
2.3.1	Bitmap	17
2.3.2	DefaultValues	17
2.3.3	HashFunction	17
2.3.4	Other	18
2.4	Dettagli tecnici	19
2.4.1	AsyncTask	19
2.4.2	SharedPreferences	19

3	Servlet	21
3.1	Introduzione alla servlet	21
3.2	Database	22
3.3	Servizi	23
4	Libreria	25
4.1	Introduzione alla libreria	25
4.2	PEST (Parameter Estimation by Sequential Testing)	26
4.3	Studio qualitativo dell'algoritmo	27
4.4	Implementazione dell'algoritmo	29
4.4.1	PART ONE	33
4.4.2	PART TWO	34
4.5	Testing dell'algoritmo	37
A	AsyncTask	45
Bibliografia		49
Sitografia		51

Capitolo 1

Introduzione generale

1.1 Ambliopia

Il termine **ambliopia** deriva dal punto di vista etimologico dalla composizione in lingua greca di ὄψ (vista) e ἀργλύς (pigra, debole). L'oftalmologia, che è la branca della medicina dell'apparato visivo, definisce l'ambliopia come l'alterazione della visione dello spazio; la motivazione è da ricercarsi in un'alterata trasmissione del segnale nervoso tra occhio e cervello per cui quest'ultimo privilegia un occhio a causa della ridotta acuità visiva dell'altro. Secondo statistiche dell'Humanitas Research Hospital [Hum17] essa è una condizione che interessa il 4% della popolazione mondiale e per le conoscenze attuali può essere trattata con successo solo se riconosciuta entro i primi 5-6 anni di vita. Data la piccola età, raramente i segnali/sintomi sono riferiti dal paziente e per questo motivo si raccomanda la prevenzione anche in assenza di sintomi attraverso visite oculistiche e screening fin dai primi anni di vita del bambino.

In caso di ambliopia contratta bisogna passare al più presto al trattamento in collaborazione con gli ortottisti. Per curarla vi sono diversi trattamenti ma tutti fondati sullo stesso principio: quello di far "sforzare" maggiormente il cosiddetto occhio pigro. Per raggiungere questo scopo tra le varie tecniche annoveriamo l'occlusione diretta con bande adesive che può avvenire o direttamente sull'occhio o con filtri semitransparenti posti sugli occhiali come in figura 1.1.



Figura 1.1. Occlusione tramite filtri semitrasparenti (patching)

Ulteriori tecniche riguardano la penalizzazione ottica con l'uso di supporti fisici. Un esempio sono gli occhiali con filtri di Bangerter, dove le lenti hanno gradi differenti di opacizzazione per l'inibizione dell'occhio sano e la stimolazione di quello ambliope.

Oltre alle tecniche fisiche esiste anche la penalizzazione farmacologica, ovvero la somministrazione di collirio a base di atropina nell'occhio dominante. Questa sostanza dilata la pupilla rendendo così più difficoltosa la vista dall'occhio sano e stimolando conseguentemente l'occhio pigro ad attivarsi.

1.2 Stereoacuità

La **stereoacuità visiva** anche detta percezione della profondità, è la più piccola differenza di profondità che può essere percepita tramite la visione binoculare tra due oggetti visibili. Essa è una delle sottocategorie della cosiddetta "acuità di allineamento o localizzazione": l'acutezza visiva dovuta al minimo spostamento spaziale percepibile tra due figure.

La misura dell'acutezza stereoscopica ha come unità di misura i secondi d'arco (arcsec). Il suo multiplo, il minuto d'arco, indica la dimensione di un oggetto in angoli. Questa espressione della dimensione di un oggetto è utile in quanto la sua dimensione percepita cambia a seconda della distanza dell'osservatore dall'oggetto stesso. Un minuto d'arco sono 60 secondi d'arco e un grado sono 60 minuti d'arco. Quindi un grado è composto da 3600 secondi d'arco.

$$1' = 60'', \quad 1^\circ = 60' \Rightarrow 1^\circ = 3600''$$

L'acutezza stereoscopica è la disparità minima oltre la quale non si produce alcun effetto stereoscopico. Più è piccolo l'angolo α e maggiore sarà l'acuità visiva secondo la legge $\tan(\alpha) = h/d$ in cui α è in funzione della distanza della mira e dell'altezza. Valori pari a circa 15-30 secondi d'arco, ossia $0.004^\circ - 0.008^\circ$, sono ritenuti eccellenti.

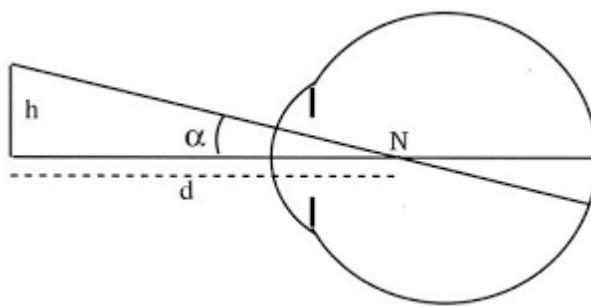


Figura 1.2. Angolo α dell'acuità visiva

Storicamente si sono delineate diverse tipologie di test che permettono di misurare la stereoacuità; il primo test fisico per la misurazione della stereoacuità è stato svolto da Howard-Dolman nel 1919.

Il test, mostrato in figura 1.3, è così strutturato:

- Di fronte all’osservatore viene posto un piolo nero (A) ad una distanza z .
- Un altro piolo nero (B) viene posto poco più avanti al primo, ad una distanza dz che viene maneggiata finché B non viene percepito come il più vicino possibile a A.
- A questo punto la distanza dz viene convertita dall’unità di misura di lunghezza utilizzata all’unità di misura angolare di disparità binoculare (la differenza del paralasse binoculare) tramite la formula:

$$d\gamma = cadz/z^2 \quad (1.1)$$

dove c è una costante (3437.75, cioè 1 radiante in arcminuti), a è la distanza interoculare dell’osservatore (la distanza tra occhio sinistro e occhio destro), dz è la distanza minima della stereoacuità rilevata e z la distanza tra gli occhi e il primo piolo A di riferimento.

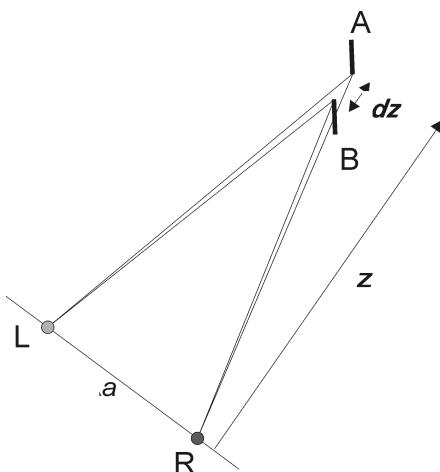


Figura 1.3. Rappresentazione grafica del test di Howard-Dolman

1.3 Anaglifi

Ora che abbiamo analizzato il funzionamento del test di stereoacuità in maniera fisica sorge una grossa problematica: per poter svolgere lo stesso test su schermi di dispositivi elettronici (cellulari nel nostro caso), che sono bidimensionali e virtuali, è necessaria un'astrazione per ottenere la tridimensionalità richiesta. Questa illusione la si ottiene con l'artificio degli anaglifi.

Un **anaglifo** è un'immagine stereoscopica che permette di avere un'illusione di realtà tridimensionale tramite l'uso di appositi occhiali dotati di due filtri di colore complementare tra loro (solitamente rosso, un colore additivo, e ciano, un colore sottrattivo). La visione binoculare permette all'uomo la visione su tre dimensioni. Questi occhiali usando i filtri permettono all'occhio di vedere solo parzialmente l'immagine completa, nascondendo la parte che viene schermata dalla lente.

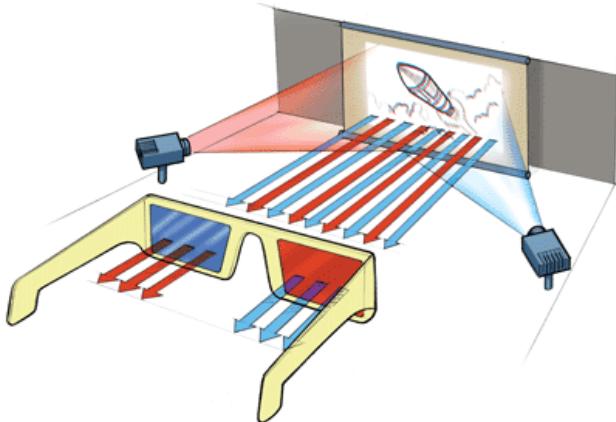


Figura 1.4. Anaglifi

L'idea alla base per ottenere lo stesso effetto tridimensionale è di avere due immagini del medesimo soggetto riprese alla stessa distanza ma scostate lateralmente con uno scarto pari alla distanza binoculare.

1.4 Il progetto 3D4AMB



Figura 1.5. Logo

Il progetto **3D4AMB**¹ [1] è nato con l’obiettivo di sviluppare un sistema basato sul 3D per la diagnosi e il trattamento dell’ambliopia nei bambini piccoli. Utilizzando la tecnologia 3D è possibile mostrare immagini diverse all’occhio sano e a quello pigro, permettendo così di diagnosticare l’ambliopia in maniera più semplice e trattarla attraverso giochi interattivi e attività di intrattenimento. I punti di forza principali del progetto 3D4AMB sono:

- **Basso costo:** i prodotti devono avere costi relativamente bassi, accessibili alle famiglie. I componenti sono infatti acquistabili presso negozi e supermercati aperti ai privati cittadini.
- **Ad uso domestico.** I sistemi sviluppati devono poter essere installati ed utilizzati anche a casa. Questo eliminerebbe una parte dei tragitti casa-clinica ed i tempi legati ad essi. In opportune condizioni, le terapie definite dai medici potranno quindi coprire periodi prolungati di erogazione ed evitare quindi la necessità di contatti frequenti tra medico e paziente.

Per quanto riguarda i software di diagnosi, una loro versione opportunamente semplificata potrebbe aiutare le scuole e il personale scolastico ad effettuare una prima, basilare ed effettivamente precoce diagnosi nei casi più semplici, in modo da avvisare i genitori che potranno accompagnare il figlio ad una prima visita oculistica.

- **Facilmente estensibili.** I diversi software sviluppati dovranno essere facilmente riutilizzabili ed estensibili con nuove funzionalità.

¹3D for the Diagnosis and Treatment of Amblyopia

Il contesto in cui è inserito questo progetto è **SE4MED**² [2], un laboratorio dove si sviluppa software per dispositivi medici usando diverse tecnologie. In esso sono poste un insieme di iniziative e progetti sulla diagnosi e trattamento dell’ambliopia.

1.5 Il nostro apporto

All’interno di questo progetto, ci siamo concentrati sullo sviluppo e la migliaia di un’applicazione di supporto ai dottori per effettuare test sull’ambliopia ai bambini all’interno di scuole o incontri solo con l’ausilio di un cellulare e di occhialini anaglifici (rossi/blu). Nel caso in cui si rilevassero disparità visive elevate si potrà approfondire l’analisi con ulteriori strumenti e in caso affermativo iniziare un trattamento per la malattia.

Tecnicamente il nostro contributo può essere suddiviso su più fronti:

- Sviluppo di un’applicazione Android, base del nostro progetto arricchita di tutte le funzioni per permettere il test e i salvataggi dei risultati.
- Aggiunta di ulteriori servizi in una servlet condivisa tra vari progetti di 3D4AMB per estendere le funzionalità dell’applicazione e il dialogo di quest’ultima con un database centrale presente sul server di SE4MED.
- Implementazione di un nuovo algoritmo per la successione di immagini e il calcolo della disparità chiamato PEST³. Esso è inserito all’interno di una libreria Java usata dalla nostra applicazione ma disponibile anche per altri progetti futuri. La principale finalità di questo nuovo algoritmo è di raggiungere, con un basso numero di prove da sottoporre in un test, il valore finale della disparità visiva del paziente. Oltre all’implementazione ci siamo anche occupati del testing di questo algoritmo per verificarne il corretto funzionamento.

Nei prossimi capitoli tratteremo di questi tre punti.

²Software Engineering for Medical Devices

³Parameter Estimation by Sequential Testing

Capitolo 2

Applicazione

2.1 Introduzione all'applicazione

L'applicazione è la base principale del progetto, è composta da varie attività che si occupano di fornire funzioni e interfaccia all'utente. Nel nostro caso un dottore, il quale potrà gestire i propri pazienti, eseguire i test e salvare i risultati. L'applicazione è stata sviluppata in Java usando l'ambiente di sviluppo Android Studio [3], basandosi sulle API 23 in modo da rendere possibile l'esecuzione dell'applicazione su un maggior numero di dispositivi, compresi i meno recenti.

Package	com.unibg.app3dsat (API 23)
Tools	Android Studio

Tabella 2.1. Dettagli applicazione

L'applicazione si interfaccia con una servlet (capitolo 3) scritta in Java, la quale fornisce una serie di servizi consentendo di leggere e modificare un database centrale; proprio per questo motivo è necessaria una connessione a internet attiva per poter usare l'applicazione. L'applicazione si serve inoltre di una libreria (capitolo 4) che abbiamo modificato introducendo un nuovo algoritmo chiamato PEST per il calcolo della disparità durante il test.

Nei prossimi paragrafi verranno analizzate le attività che compongono l'applicazione; infine verranno presentati alcuni dettagli tecnici e tecniche usate.

2.2 Main activities

2.2.1 Login

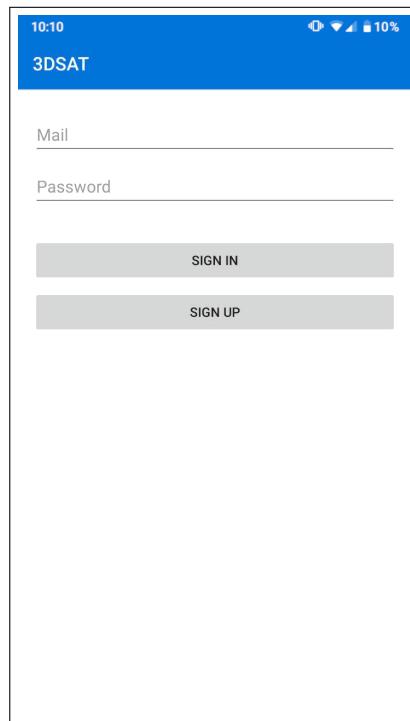


Figura 2.1. LoginActivity.java

All'apertura dell'applicazione verrà presentato il form per il login del dottore. Le credenziali d'accesso possono essere ottenute dal dottore registrandosi su se4med.unibg.it/home, sito raggiungibile facilmente cliccando il pulsante *sign up*. In caso di digitazione errata della password o errori di rete verrà visualizzato un messaggio di errore nell'interfaccia.

2.2.2 Main Doctor

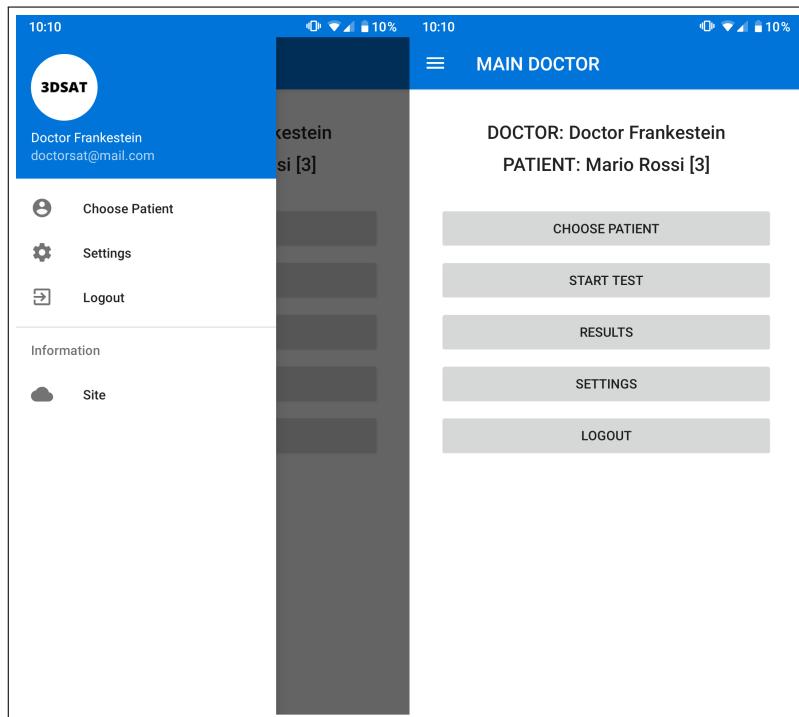


Figura 2.2. MainDoctor.java

Dopo aver effettuato l'accesso correttamente, verrà presentato al dottore il menu principale compreso di navbar che riporta i dati del dottore. Sempre nella navbar è presente un collegamento alla versione web tramite un link al sito se4med.unibg.it/home. Nell'intestazione troviamo anche il nome del paziente attualmente selezionato con il relativo ID; al primo accesso nell'applicazione non verrà visualizzato nulla, questo perché non è stato ancora selezionato/creato nessun paziente. È necessario quindi andare a selezionare/creare il paziente nell'apposita attività.

Le sezioni per iniziare il test e per visualizzare i risultati sono accessibili soltanto se un paziente è stato correttamente selezionato. Sarà inoltre possibile modificare le impostazioni dell'applicazione nell'apposita attività oppure effettuare il logout dal menu o dalla navbar e tornare al login form.

2.2.3 Main Patient

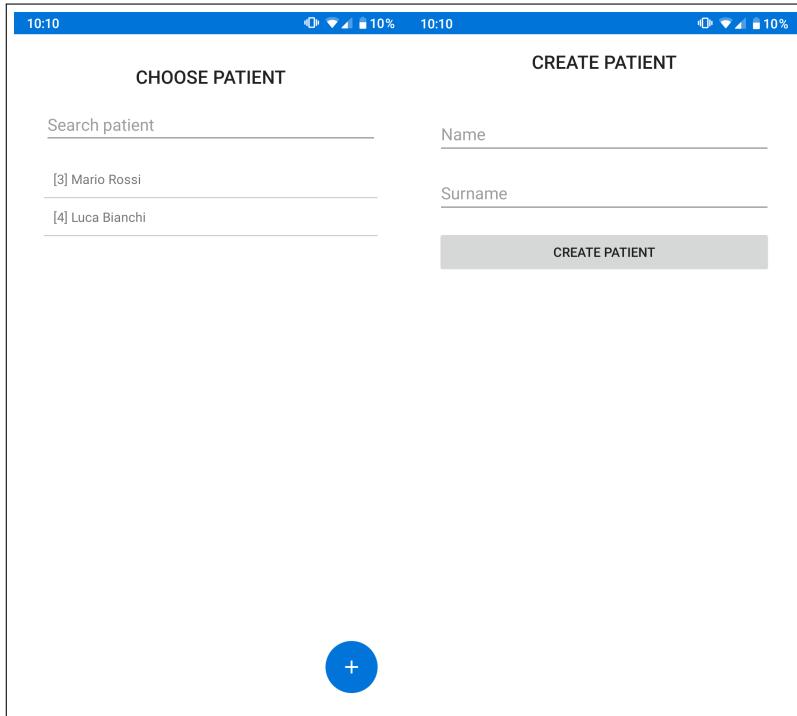


Figura 2.3. MainPatient.java, CreatePatient.java

A questa sezione si accede cliccando il pulsante *choose patient* nel menu del dottore. Nell’attività *MainPatient.java* verrà scaricata automaticamente dalla servlet la lista di tutti i pazienti gestiti dal dottore stampati in questo formato “[ID] Nome Cognome”. È presente una barra di ricerca per velocizzare l’individuazione e la selezione di un paziente.

Selezionando un qualsiasi paziente si ritorna nel menu principale del dottore e verrà visualizzato il paziente appena selezionato; sarà inoltre possibile iniziare il test o vedere i risultati dei test precedentemente ottenuti dal paziente.

Cliccando sul bottone blu si passa all’attività *CreatePatient.java* dove è possibile creare un paziente compilando con nome e cognome (ID automaticamente generato dalla servlet con una numerazione automatica incrementale). Dopo la creazione, il paziente appena creato verrà automaticamente selezionato ritornando nel menu del dottore.

2.2.4 Test

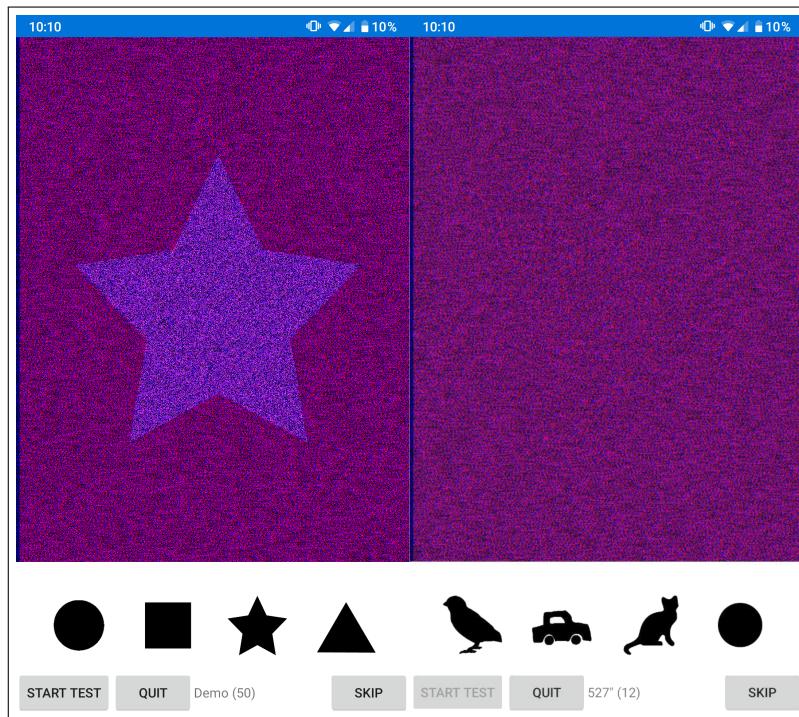


Figura 2.4. Test.java

All’apertura del test saranno visualizzate immagini in versione demo (figura 2.4 a sinistra), come si può notare dalla scritta in basso, per far prendere confidenza al paziente del funzionamento del test e dei tasti. In questa modalità le figure sono facilmente visibili anche senza gli occhialini in aiuto del paziente.

Nella parte centrale vengono rappresentate le immagini contenenti le figure presenti nella matrice dei puntini rossi e blu (esistono diversi set di figure selezionabili nelle impostazioni). Nella barra sottostante vi sono le quattro figure candidate a essere soluzione, ovvero uguali all’immagine soprastante; il paziente dovrà selezionare quella che ha visto nell’immagine sopra per proseguire il test con una nuova immagine.

In basso, tra i pulsanti, è descritta la fase del test: la modalità demo (figura 2.4 a sinistra) oppure il livello di disparità dell’immagine visualizzata (figura 2.4 a destra).

I tasti in grigio sono:

- **START TEST**: inizia il test vero e proprio saltando la demo mode; verrà quindi rappresentata l'immagine con il livello di disparità più alto, l'abbinamento più semplice da risolvere (modificabile dalle impostazioni, di default è 12).
- **QUIT**: uscita immediata dal test; viene chiesto se si vuole salvare lo stato del test interrotto nei risultati.
- **SKIP**: sostituisce l'immagine attuale con un'altra generata in modo casuale, senza nessuna penalizzazione per il punteggio del test.

L'algoritmo che sceglie le immagini da visualizzare e le relative disparità è implementato nella libreria e verrà descritto in seguito.

Al termine del test i risultati saranno visualizzati brevemente in una schermata (*TestResults.java*) e saranno automaticamente caricati sulla servlet; nel caso di errore di rete verranno effettuati dei tentativi di ricaricamento dei dati. Tutti i risultati di un paziente sono sempre disponibili nell'attività *Results.java*.

2.2.5 Results

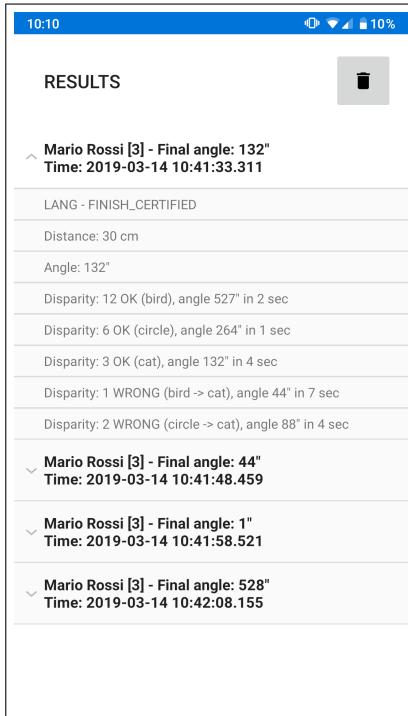


Figura 2.5. Results.java

In questa attività è presente lo storico dei risultati relativi ai singoli test effettuati dallo specifico paziente selezionato. L’elenco è in ordine cronologico e oltre al timestamp è anche indicato anche il *final angle* raggiunto dal paziente.

Espandendo un test specifico cliccandoci sopra, si potranno visualizzare ulteriori dettagli: la certificazione raggiunta dal paziente, la distanza dallo schermo con cui si è effettuato il test, l’angolo finale raggiunto dal paziente e la lista di tutti i tentativi fatti dal paziente indicanti la disparità, la coppia figura selezionata-figura giusta e in quanto tempo il paziente ha effettuato la scelta.

Tenendo premuto su un test specifico è possibile richiedere la sua eliminazione del singolo test dalla servlet, oppure, cliccando il tasto in alto a destra sull’icona del cestino, è possibile richiedere l’eliminazione di tutti i test di quello specifico paziente dalla servlet.

2.2.6 Settings

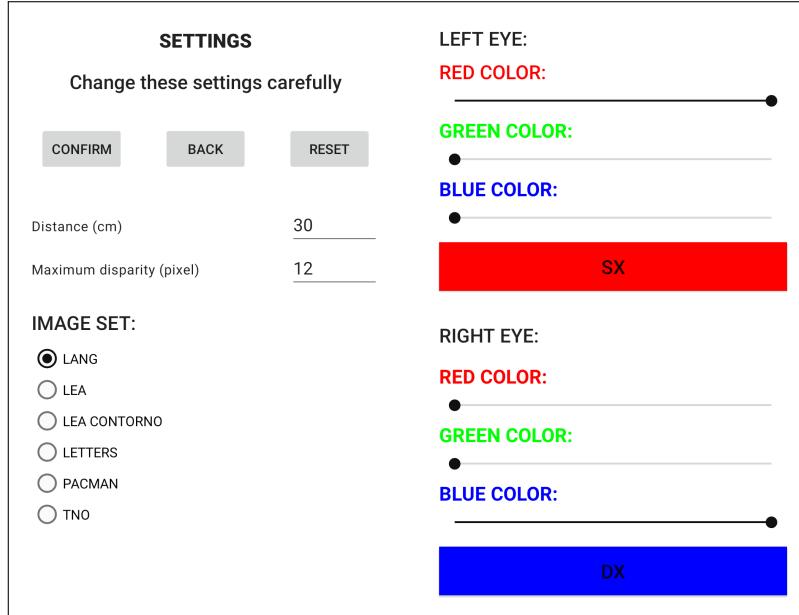


Figura 2.6. Settings.java

In questa attività è possibile modificare le impostazioni dell'applicazione. In particolare è possibile modificare la distanza dallo schermo, la disparità massima e il livello iniziale di disparità con cui si inizia il test. La disparità più piccola che bisognerà raggiungere durante il test è fissata per default a 1. In caso di errata resa dei colori dello schermo del cellulare è possibile settare manualmente i colori blu e rosso bilanciando i valori RGB.

Inoltre, è possibile modificare il set di immagini (figura 2.7) che verranno visualizzate durante il test; in ordine dall'alto al basso, da sinistra a destra sono: LANG, LEA, LEA CONTORNO, LETTERS, PACMAN, TNO.

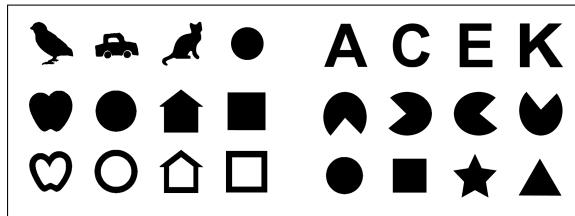


Figura 2.7. Set di immagini

2.3 Util activities

2.3.1 Bitmap

Activities: *BitmapBuilder.java, BitmapParallelBuilder.java, SingleBitmapBuilder.java*

Sono tutte classi che lavorano in background che si occupano di creare la matrice formata dai puntini blu/rossi che andranno a comporre l'immagine da visualizzare durante il test.

2.3.2 DefaultValues

Activity: *DefaultValues.java*

In questa classe sono dichiarate tutte le costanti statiche che verranno usate nel codice dalle altre attività e tutti i valori di default delle varie variabili usate nell'applicazione.

Degne di nota sono queste due costanti:

```
1 public static final String AUTHORITY = "se4med.unibg.it";  
2 public static final String STEREOTEST = "StereoTest";
```

La prima indica l'indirizzo su cui è ospitata la servlet, costante usata durante ogni connessione presente nell'applicazione. La seconda indica l'ID dell'applicazione usato per identificare il nostro progetto tra tutti quelli presenti in 3D4AMB e SE4MED.

2.3.3 HashFunction

Activity: *HashFunction.java*

Classe contenente metodi necessari per convertire la password inserita dal dottore durante il login in SHA-256. Viene quindi salvato in Shared Preferences il corrispondente codice SHA-256 generato dalla password che trasmesso come parametro alla servlet a ogni richiesta di un servizio, compreso il login. In questo modo è possibile oscurare la password durante la connessione con la servlet evitando di passare la password in chiaro.

Il codice della classe HashFunction è il seguente:

```
1 public class HashFunction {  
2     public static String toSha256(String password) throws  
3         NoSuchAlgorithmException {  
4         MessageDigest md = MessageDigest.getInstance("SHA-256");  
5         md.update(password.getBytes());  
6         return bytesToHex(md.digest());  
7     }  
8     private static String bytesToHex(byte[] bytes) {  
9         StringBuilder result = new StringBuilder();  
10        for (byte byt : bytes)  
11            result.append(Integer.toString((byt & 0xff) + 0x100,  
12                16).substring(1));  
13        return result.toString();  
14    }  
}
```

2.3.4 Other

Activities: *Group.java, ListViewAdapter.java, MyExpandableListAdapter.java*

Classi di supporto nella creazione dei menu, delle liste o degli elenchi presenti nell'applicazione.

Activity: *Patient.java*

Attività di supporto durante la composizione della tripla "*nome, cognome, ID*" del paziente.

2.4 Dettagli tecnici

2.4.1 AsyncTask

Ogni connessione verso la servlet viene effettuata con un task asincrono (AsyncTask) in modo da evitare il blocco dell'applicazione. Tutte le connessioni seguono un pattern ricorrente e molto simile tra loro; è possibile consultare l'appendice A per il codice completo.

2.4.2 SharedPreferences

Quest'applicazione usa SharedPreferences per salvare i dati riguardanti il dottore, il paziente e le impostazioni.

Listing 2.1. Porzione di codice presa da DefaultValues.java

```
1  /**
2   * SPDOCTOR
3   */
4  public static final String SPDOCTOR = "SPDoctor";
5  public static final String ACTUAL_DOCTOR_MAIL = "actual_doctor_mail";
6  public static final String ACTUAL_DOCTOR_PASSWORD = "actual_doctor_password";
7  public static final String ACTUAL_DOCTOR_NAME = "actual_doctor_name";
8  public static final String ACTUAL_DOCTOR_SURNAME = "actual_doctor_surname";
9
10 /**
11  * SPPATIENT
12 */
13 public static final String SPPATIENT = "SPPatient";
14 public static final String ACTUAL_PATIENT_NAME = "actual_patient_name";
15 public static final String ACTUAL_PATIENT_SURNAME = "actual_patient_surname";
16 public static final String ACTUAL_PATIENT_ID = "actual_patient_id";
17
18 /**
19  * SPSETTINGS
20 */
21 public static final String SPSETTINGS = "SPSettings";
22 public static final String PREF_DISTANCE = "pref_distance";
23 public static final String PREF_MAXDISPARITY = "pref_maxdisparity";
24 public static final String PREF_MINDISPARITY = "pref_mindisparity";
25 public static final String PREF_IMAGESET = "pref_imageset";
```

```
26 public static final String PREF_OFFSET = "pref_offset";
27 public static final String PREF_NCRR_TO_NEXLEVEL = "pref_ncorronextlevel";
28 public static final String PREF_NERR_TOSTOPTEST = "pref_nerrtostoptest";
29 public static final int DEFAULT_DISTANCE = 30;
30 public static final int DEFAULT_MAXDISPARITY = 12;
31 public static final int DEFAULT_OFFSET = 1;
32 public static final int DEFAULT_NCRR_TONEXTLEVE = 1;
33 public static final int DEFAULT_NERR_TOSTOPTEST = 3;
34 public static final ImageShape.ImageSet DEFAULT_IMAGESET =
    ImageShape.ImageSet.LANG;
35 public static final int POSSIBLECHOICES = 4; // Possible choices in Test
```

Capitolo 3

Servlet

3.1 Introduzione alla servlet

La servlet ospitata all’indirizzo *se4med.unibg.it/home* è condivisa tra tutti i progetti universitari e mette a disposizione una serie di servizi atti sia all’autenticazione delle varie applicazioni che al salvataggio di dati in un database centrale. La servlet è scritta in Java, noi abbiamo usato l’ambiente di sviluppo Eclipse Maven [4] per implementare nuovi metodi che la nostra applicazione richiedeva. Oltre alla servlet abbiamo lavorato anche sul database usando MySQL [5].

Servlet	<i>se4med.unibg.it/home</i>
Tools	Eclipse Maven, MySQL

Tabella 3.1. Dettagli servlet

Abbiamo anche creato un account di prova su *se4med.unibg.it/home* per un dottore dove effettuare tutti i test necessari:

Account	<i>doctorsat@mail.com</i>
Password	<i>doctorsat</i>

Tabella 3.2. Credenziali dell’account di prova

3.2 Database

Il database connesso alla servlet è molto vasto, le tabelle usate dal nostro progetto sono quelle rappresentate in figura 3.1.

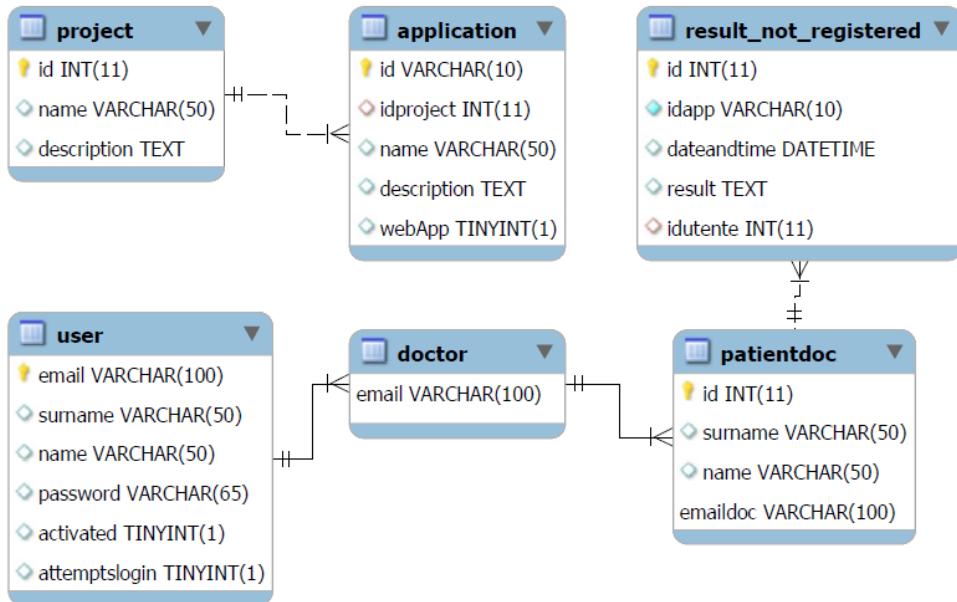


Figura 3.1. Database della servlet

- Tabella **project**: elenca tutti i progetti presenti nel database, tra i quali il nostro denominato *Stereo Acuity Test*.
- Tabella **application**: elenca tutte le applicazioni create, riferite a un particolare progetto; la nostra applicazione ha come *idapp* il valore *StereoTest*.
- Tabella **user**: elenca tutti gli utenti che hanno effettuato la registrazione.
- Tabella **doctor**: elenca tutti i dottori registrati.
- Tabella **patientdoc**: elenca tutti i pazienti gestiti da uno specifico dottore.
- Tabella **result_not_registered**: elenca tutti i risultati dei test effettuati dai pazienti; i dati sono salvati in formato JSON¹ [6].

¹JavaScript Object Notation

Nella servlet, per scrivere il codice necessario per interrogare e modificare il database, ci siamo serviti di JOOQ² [7], una libreria software leggera per il database-mapping in Java che fornisce un linguaggio specifico di dominio per costruire query da classi generate da uno schema di database.

3.3 Servizi

Nella servlet abbiamo implementato questi nuovi servizi usati dall'applicazione:

- Servizio **authenticate_doctor_ns_action**: autenticazione del dottore, usata dall'applicazione nel login form iniziale.
- Servizio **getpatientdoclist_action**: ritorna la lista dei pazienti gestiti dal dottore specifico in formato "*nome, cognome, id*".
- Servizio **createpatientdoc_action**: crea un paziente con nome, cognome e id autogenerato e lo associa al dottore.
- Servizio **deletepatientdoc_action**: cancella lo specifico paziente richiesto dal dottore.
- Servizio **storeresults_not_registered_action**: salvataggio del risultato finale del singolo test effettuato dal paziente.
- Servizio **getresults_not_registered_action**: ritorna la lista di tutti i risultati dei test di uno specifico paziente associato al dottore.
- Servizio **deleteresults_all_action**: cancella tutti i risultati dei test di uno specifico paziente associato al dottore.
- Servizio **deleteresult_single_action**: cancella il singolo test selezionato di uno specifico paziente associato al dottore.

²Java Object Oriented Quering

Le risposte all’interrogazione della servlet sono in formato JSON e nell’applicazione è possibile estrarre il contenuto e l’informazione desiderata. Di seguito sono elencate tutte le interfacce usate per l’interrogazione e la modifica del database usando JOOQ con i rispettivi parametri passati durante la richiesta:

Listing 3.1. Interfacce per l’interrogazione e la modifica del database

```
1 String getNameSurnameDoctor(String emailldoc, String password, String idapp);  
2  
3 ArrayList<String> getPatientDocList(String emailldoc, String password, String  
4 idapp);  
5  
6 int createPatientDoc(String emailldoc, String password, String name, String  
7 surname, String idapp);  
8  
9 int deletePatient(String emailldoc, String password, String idpatient, String  
10 idapp);  
11  
12 ArrayList<String> getResultsNotRegistered(String emailldoc, String password, String  
13 idpatient, String idapp, Timestamp datetime,  
14 String result);  
15  
16 int deleteResultsAll(String emailldoc, String password, String idpatient, String  
17 idapp);  
18  
19 int deleteResultSingle(String emailldoc, String password, String idpatient,  
20 String idapp, String results);

---


```

Capitolo 4

Libreria

4.1 Introduzione alla libreria

La libreria per lo *Stereo Acuity Test* è stata modificata implementando il PEST¹, un nuovo algoritmo di calcolo dei livelli di profondità dell'immagine che permette in pochi passaggi di trovare la profondità massima raggiungibile dal paziente. Questa libreria è usata dall'applicazione per determinare la sequenza di immagini con la relativa disparità mostrate durante il test; tuttavia essa potrà essere usata in futuro da qualsiasi altra applicazione che necessiterà dell'algoritmo PEST. Per modificare la libreria scritta in Java abbiamo usato l'ambiente di sviluppo Eclipse Maven [4].

Library	lib3dsat_pest.jar
Tools	Eclipse Maven

Tabella 4.1. Dettagli libreria

Per comprendere l'algoritmo ci siamo basati sia su un estratto di Alex Pentland "Maximum likelihood estimation: The best PEST" [Pen80] che su un articolo di Neil Carter "Parameter Estimation by Sequential Testing" [Car17].

¹Parameter Estimation by Sequential Testing

4.2 PEST (Parameter Estimation by Sequential Testing)

L'**algoritmo PEST** si basa su una procedura descritta da M. M. Taylor e C. Douglas Creelman in "*PEST: Efficient Estimates on Probability Functions*" [TC67]. In esso sono definite un insieme di regole da seguire per settare e impostare la difficoltà di un determinato processo, con lo scopo di riuscire a rilevare il più velocemente possibile il livello a cui le performance tendono in maniera predefinita. In maniera formale si sostiene che: dati dei segnali con determinate proprietà (variabili indipendenti) si deve ricercare e regolare la precisione delle performance da sottoporre a seconda dei risultati ottenuti (variabili dipendenti).

Nel caso di Taylor e Creelman il test riguardava alcuni partecipanti sui quali furono testate le capacità di ascolto di una varietà di suoni. Il test veniva considerato corretto con una soglia del 25%, ovvero doveva essere riconosciuto dai partecipanti entro il 75% delle volte che veniva eseguito. L'obiettivo era capire la velocità del ragionamento mentale che portava a riconoscere il suono come risposta a stimoli ripetuti. Il test venne svolto per risposta diretta sia per selezione tra un numero predefinito di opzioni. A seconda dell'esito della risposta e del tempo di risposta (indice di accuratezza e precisione) il livello veniva cambiato per avere un processo più difficile o viceversa. Il test si concludeva quando una serie di prove rimaneva costante su un certo livello che venne chiamato "step" o "block".

L'algoritmo PEST si rivelò molto efficiente nel riuscire a individuare il livello di performance dei partecipanti con il minimo numero di prove possibile. La sua efficienza è dovuta alle regole che governano la scelta del livello successivo da sottoporre sulla base degli esiti delle risposte precedenti.

Gli studi di Taylor e Creelman del 1967 furono portati avanti da diversi studiosi e furono sviluppate diverse versioni del PEST tra le quali annoveriamo quella di Findlay nel 1978, la cui formula è presente in figura 4.1.

$$m_n = \max \Pr[x \text{ is } 50\% \text{ point} | (m_1, r_1), (m_2, r_2) \dots (m_{n-1}, r_{n-1})]$$

Figura 4.1. Formula di Findlay

La lettera m rappresenta la variabile indipendente da misurare/impostare e come pedice è indicata l’i-esima misurazione svolta. La r corrisponde al risultato della prova e può essere o +1 se corretta o -1 se errata. Il valore 50% indica la verosimiglianza che si vuole ottenere.

4.3 Studio qualitativo dell’algoritmo

In questo paragrafo ci addentreremo negli studi di Taylor e Creelman e in particolare sull’analisi del loro test per certificare i vantaggi che questo algoritmo può apportare.

Il PEST monitora le performance dei partecipanti con una semplice operazione: il calcolo del rapporto tra il numero di risposte corrette date da un partecipante e il totale delle prove; in questo modo si ottiene il punteggio (**score**) del partecipante che varierà tra zero (tutte risposte sbagliate) e 1 (tutte risposte corrette). Esso viene confrontato con la precisione (**accuracy**) richiesta dal test e da questo confronto si può prendere una scelta sul livello successivo da sottoporre. La precisione è un parametro molto rilevante che è già preimpostato prima del test: un’alta precisione implica una maggior difficoltà nello svolgere il test e viceversa. Un altro parametro è la deviazione (**deviation limit**), che previene che ci siano cambiamenti troppo repentini tra un livello e il successivo. Quest’ultima verrà molto utile nei casi particolari come per esempio nel primo livello dove a seconda dell’esito si avrà uno score massimo=1 o minimo=0.

Per studiare qualitativamente l’algoritmo useremo dei **grafici** che permettano di analizzare la relazione che intercorre tra l’andamento delle risposte corrette e il susseguirsi delle prove con i relativi livelli di difficoltà (rispettivamente fissati sull’asse delle ordinate e delle ascisse).

Nella prima coppia di grafici (vedi figura 4.2) analizziamo come varia l’algoritmo PEST in funzione della deviazione standard. La banda verde indica il range di valori nel quale non è richiesto un cambiamento di difficoltà della prova. Con i punti rossi e blu indichiamo rispettivamente i casi limite in cui il partecipante risponde rispettivamente in maniera errata o corretta a tutte le prove sottoposte. Confrontando i grafici notiamo che un aumento della deviazione standard relativamente piccolo diminuisce il range in cui è richiesto al PEST un cambio di livello difficoltà. In altri termini la deviazione standard è una grandezza direttamente proporzionale

alla stabilità con quest'ultima intesa come mantenimento costante del livello tra step successivi.

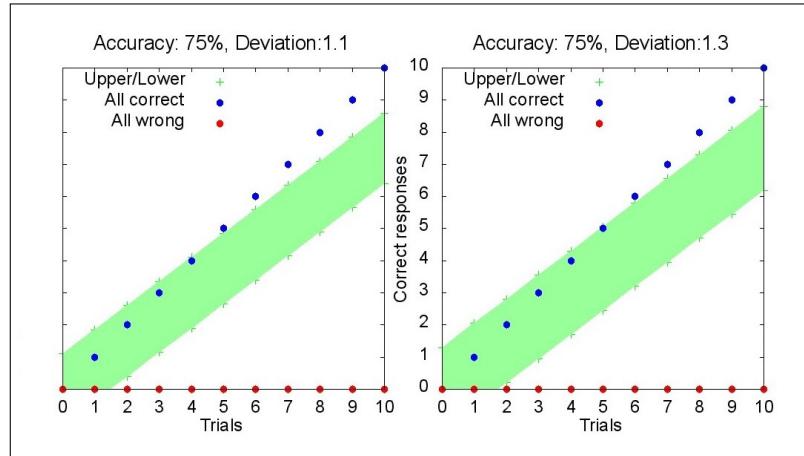


Figura 4.2. Grafici con precisione 75%

Negli ultimi due grafici (vedi figura 4.3) invece valutiamo gli effetti causati da un aumento del 10% di precisione. La banda in cui è richiesta una difficoltà costante risulta essere più aderente alle risposte corrette. In genere un aumento di accuratezza provoca nel caso di risposta corretta una nuova domanda molto più difficile e viceversa in caso di risposta errata una nuova domanda molto più facile. Sempre in termini di stabilità come precedentemente definita, possiamo affermare che la precisione risulta essere inversamente proporzionale alla stabilità.

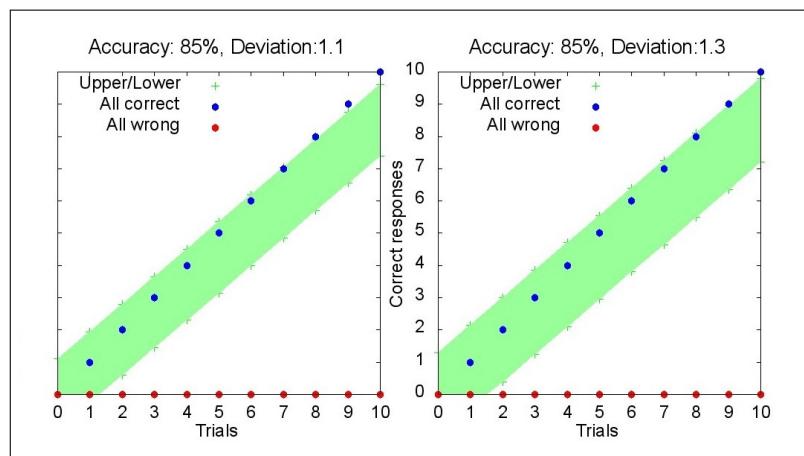


Figura 4.3. Grafici con precisione 85%

4.4 Implementazione dell’algoritmo

Abbiamo implementato due algoritmi di PEST entrambi testati con i relativi casi di test. Il primo algoritmo si trova nella classe *PestDepthCertifierOld* e segue di pari passo le regole definite da Taylor e Creelman nel loro saggio. Tuttavia, nella nostra applicazione è stato implementato un’evoluzione di tale algoritmo con accorgimenti e modifiche funzionali atte all’adattarsi ai nostri requisiti. Questo algoritmo si trova nella classe *PestDepthCertifier* e verrà gradualmente descritto in questo capitolo.

Ad ogni livello di disparità è associato un angolo che indica la profondità dell’immagine blu/rossa sullo schermo.

Osservazione 1 I numeri dei livelli sono inversamente proporzionali alla profondità dell’immagine e quindi alla difficoltà del livello. Il livello con maggiore profondità, quindi il più difficile da superare, è il livello 1.

Un insieme definito da due variabili *leftLimit* e *rightLimit*, che indicano rispettivamente il limite sinistro e destro di tale insieme, è necessario per racchiudere all’interno il valore corrente della disparità di un determinato livello (variabile *currentDepth*).

Osservazione 2 $\text{leftLimit} > \text{currentDepth} > \text{rightLimit}$

Per istanziare la classe **PestDepthCertifier** (codice 4.1) è richiesto un solo argomento: *initDepth* = *maxDepth* che è il livello iniziale, ossia il più semplice da superare che è anche il massimo livello nel test. Il livello iniziale è impostato a 12 per default ma può essere cambiato nelle impostazioni dell’applicazione dal dottore; il livello minimo, quello più complesso da raggiungere, è fissato a 1. Secondo quanto visto nell’ultima osservazione viene inizializzato l’insieme ponendo *leftLimit* = *initDepth* e *rightLimit* = 1.

Viene istanziatato anche un’array chiamato *vector* con dimensione uguale al numero di livelli del test (*initDepth*) che permetterà di salvare le risposte date dal paziente nel seguente modo: inizialmente i suoi elementi saranno posti tutti a zero (ad esempio con *initDepth* = 12 avremo [0,0,0,0,0,0,0,0,0,0]). Ogni posizione dell’array corrisponde ad un livello: le risposte corrette sono registrate sommando un 1 nella specifica posizione dell’array, mentre le risposte errate sono salvate sottraendo un 1.

La variabili *chance* e *weight* verranno spiegate in seguito.

Listing 4.1. Classe PestDepthCertifier

```
1 public PestDepthCertifier(int initDepth) {
2     certifierStatus = new CertifierStatus();
3
4     if (initDepth >= 1)
5         certifierStatus.currentDepth = initDepth;
6     else
7         throw new IllegalArgumentException();
8
9     maxDepth = initDepth;
10    leftLimit = initDepth;
11    rightLimit = 1;
12    chance = 1;
13    vector = new int[initDepth];
14    weight = 1;
15    partone = true;
16
17    certifierStatus.currentResult = CONTINUE;
18 }
```

Prima dell'avvio del test si avrà una demo del livello con disparità *initDepth* per far prendere familiarità con i comandi ai pazienti e successivamente inizierà il test vero e proprio.

Il metodo **computeNextDepth** (codice 4.2) di tipo void riceve come argomento l'esito della risposta data dal paziente ad un determinato livello e procede a computare il livello successivo da sottoporre al paziente. Nel caso in cui si sia raggiunto il livello finale o l'algoritmo riconosca il livello di performance del paziente, si procede a stampare la certificazione.

L'algoritmo è composto da due macro-fasi in successione.

La prima fase (**PART ONE**) ha lo scopo di individuare con il minor numero di prove possibile un range di due valori in cui è molto alta la probabilità che vi si trovi il livello di performance del paziente.

La seconda fase (**PART TWO**) partendo dal range trovato nella fase precedente permette di determinare precisamente il livello del paziente.

Listing 4.2. Método computeNextDepth

```
1 void computeNextDepth(DepthCertBase.Solution solution) {
2     int savedCurrentDepth = certifierStatus.currentDepth;
3
4     // PART ONE
5     // Quickly and approximately identify the range in which the level can be
6     // contained
7     if (partone && (solution != DepthCertBase.Solution.NULL)) {
8         System.out.print("[PART ONE]");
9
10        if (solution == DepthCertBase.Solution.WRONG && chance > 0 &&
11            certifierStatus.currentDepth == maxDepth) {
12            // First wrong attempt to maxDepth
13            chance--;
14        } else if (solution == DepthCertBase.Solution.WRONG && chance == 0
15                   && certifierStatus.currentDepth == maxDepth) {
16            // Second wrong attempt to maxDepth
17            certifierStatus.currentResult = FINISH_NOT_CERTIFIED;
18        } else if (solution == DepthCertBase.Solution.RIGHT) {
19            leftLimit = certifierStatus.currentDepth;
20
21            // Numerical rounding (Floor: round down)
22            value = ((double) leftLimit + rightLimit) / 2;
23            nextDepth = (int) (Math.floor(value));
24
25            certifierStatus.currentDepth = nextDepth;
26            vector[leftLimit - 1] += 1;
27
28            if ((leftLimit - rightLimit) == 1) {
29                partone = false;
30                nextDepth = rightLimit;
31                certifierStatus.currentDepth = nextDepth;
32
33                // Weight because the rightLimit was wrong
34                if (rightLimit != 1)
35                    weight = weight*3;
36            }
37        } else if (solution == DepthCertBase.Solution.WRONG) {
38            rightLimit = certifierStatus.currentDepth;
39
40        }
41    }
42}
```

```
38     // Numerical rounding (Ceil: round up)
39     value = ((double) leftLimit + rightLimit) / 2;
40     nextDepth = (int) (Math.ceil(value));
41
42     certifierStatus.currentDepth = nextDepth;
43     vector[rightLimit - 1] -= 1;
44
45     if ((leftLimit - rightLimit) == 1) {
46         partone = false;
47         nextDepth = rightLimit;
48         certifierStatus.currentDepth = nextDepth;
49
50         // Weight because the rightLimit was wrong
51         if (rightLimit != 1)
52             weight = weight*3;
53     }
54 }
55
56 // PART TWO
57 // Focus on the lower level identified [nextDepth = rigthLimit] until it
      reaches
58 // a +2 that certifies the level, or a -2 that shift to an easier level
59 // (nextDepth++). Added weight to avoid loop like [OK | NO | OK | NO]
60 } else if (!partone && (solution != DepthCertBase.Solution.NULL)) {
61     System.out.print("[PART TWO]");
62
63     if (solution == DepthCertBase.Solution.RIGHT) {
64         vector[nextDepth - 1] += 1;
65         certifierStatus.currentDepth = nextDepth;
66     } else if (solution == DepthCertBase.Solution.WRONG) {
67         // Subtract with weight
68         vector[nextDepth - 1] -= weight;
69         weight = weight * 3;
70         certifierStatus.currentDepth = nextDepth;
71     }
72
73     // Vector control and maxDepth control
74     if (vector[nextDepth - 1] >= 2) {
75         // Reached the certification
76         certifierStatus.currentResult = FINISH_CERTIFIED;
```

```

77         certifierStatus.currentDepth = nextDepth;
78     } else if ((vector[nextDepth - 1] <= -2) && (nextDepth < maxDepth)) {
79         // Shift the nextDepth and reset weight
80         nextDepth++;
81         weight = 1;
82         certifierStatus.currentDepth = nextDepth;
83     } else if ((vector[nextDepth - 1] <= -2) && (nextDepth == maxDepth)) {
84         // Returned to the initial maxDepth without certification
85         certifierStatus.currentResult = FINISH_NOT_CERTIFIED;
86         certifierStatus.currentDepth = nextDepth;
87     }
88 } else {
89     System.out.print("[NULL (SKIP BUTTON)]");
90     assert solution == DepthCertBase.Solution.NULL;
91 }
92 }
```

4.4.1 PART ONE

Nella prima fase, ad ogni step l’algoritmo conosce il valore di *currentDepth* e riceve l’esito della risposta data dal paziente sulla base della figura scelta:

- **RIGHT**: come conseguenza ad una risposta corretta del paziente si aggiorna il limite sinistro con il livello corrente appena superato e si calcola il livello successivo come la media tra il limite sinistro e il limite destro con arrotondamento per difetto.

$$leftLimit = currentDepth$$

- **WRONG**: come conseguenza ad una risposta errata del paziente si aggiorna il limite destro con il livello corrente appena sbagliato e si calcola il livello successivo come la media tra il limite sinistro e il limite destro con arrotondamento per eccesso.

$$rightLimit = currentDepth$$

- **NULL (SKIP)**: non si esegue nulla, verrà riproposta un’altra immagine di pari livello di disparità.

Le tre casistiche sono mutuamente esclusive e portano ad una riduzione progressiva dell’insieme. Nel momento in cui l’insieme si riduce a due livelli consecutivi ($leftLimit - rightLimit == 1$) la prima fase termina. Notiamo che ad ogni step, prima di effettuare la decisione del livello successivo, l’algoritmo salva l’esito della risposta data dal paziente nell’array *vector*.

4.4.2 PART TWO

La seconda fase ha inizio con la terminazione della prima; poniamo la variabile *nextDepth* = *rightLimit* in modo che il prossimo livello da testare sarà il limite destro. Il valore del livello *rightLimit* nell’array avrà valore -1: la motivazione risiede nel fatto che nella prima parte dell’algoritmo il limite destro viene aggiornato solo in presenza di una risposta errata e, viceversa, il limite sinistro viene aggiornato solo in conseguenza di una risposta corretta. Inoltre la variabile *weight* sarà pari a 3 avendo già sbagliato quel livello nella prima parte dell’algoritmo. Un’eccezione è quando *rightLimit* = 1, in questo caso il valore nell’array è 0 e la variabile *weight* sarà pari a 1 perchè non è stato mai commesso un errore sul livello 1.

Una serie di tre risposte corrette senza errori porterà a terminare l’algoritmo restituendo quel livello, aumentando a ogni step di 1 il valore dell’array fino a raggiungere 2 che provoca un’uscita certificata. Un eventuale errore provocherà l’esclusione di quel livello e l’inizio di un processo ricorsivo governato dalle seguenti regole:

- In caso di risposta corretta verrà sommato 1 alla posizione (*i* - 1) dell’array, corrispondente al livello *nextDepth* sottoposto al paziente.

vector[nextDepth - 1] += 1;

- In caso di risposta errata verrà sottratto *weight* alla posizione (*i* - 1) dell’array, corrispondente al livello *nextDepth* sottoposto al paziente. In particolare al primo errore la variabile *weight* avrà valore 1, mentre ad ogni errore successivo avrà valore (*weight* * 3). Facendo pesare maggiormente i successivi errori evitiamo di entrare in una situazione di loop causata dal paziente che alterna una risposta giusta a una sbagliata.

*vector[nextDepth - 1] -= (1 * weight);*

*weight = weight * 3;*

- Nel caso in cui la posizione $(i - 1)$ dell’array assumesse valore maggiore o uguale a 2, il livello $nextDepth$ viene certificato, ossia è stato superato correttamente almeno due volte.

$vector[nextDepth - 1] >= 2$

- Nel caso in cui la posizione $(i - 1)$ dell’array assumesse valore minore o uguale a -2, la variabile $nextDepth$ viene incrementata permettendo di sottoporre al paziente un livello più semplice; inoltre viene resettata a 1 la variabile $weight$.

$vector[nextDepth - 1] <= -2$

$nextDepth++;$

$weight = 1;$

Un ulteriore controllo viene effettuato sulla variabile $nextDepth$ in modo tale che non venga incrementata oltre il livello iniziale $initDepth = maxDepth$.

Per gestire la terminazione, quindi la certificazione del livello raggiunto o meno dal paziente, è necessaria la variabile $certifierStatus.currentResult$ che può assumere tre possibili stati:

- **FINISH_CERTIFIED**: l’algoritmo riconosce di aver individuato la performance (profondità) raggiunta dal paziente. La variabile $nextDepth$ conterrà il livello raggiunto dal paziente.
- **CONTINUE**: sono necessarie ulteriori prove per stabilire il livello finale.
- **FINISHED_NOT_CERTIFIED**: l’algoritmo termina il test senza riconoscere la certificazione. Avviene nei seguenti casi:

Durante la **prima fase**, se il paziente ha sbagliato consecutivamente le prime due risposte i cui livelli sono entrambi settati alla profondità iniziale; non avendo risposto a nessuna domanda nel modo corretto il test termina senza la certificazione. Il controllo avviene tramite la variabile $chance$ inizializzata a 1 che viene decrementata solo nel caso in cui venga sbagliato il livello iniziale. Quindi al secondo errore la variabile $chance$ avrà valore 0 e provocherà l’uscita non certificata.

$rightLimit == currentDepth \& chance == 0$

Durante la **seconda fase**, a seguito di numerosi errori consecutivi e livelli sempre più facili sottoposti progressivamente, il paziente non è stato in grado di saper risolvere nemmeno il livello massimo (iniziale) nonostante nella prima fase l'abbia risolto correttamente. Nessun livello può essere certificato.

(vector[nextDepth - 1] <= -2) && (nextDepth == maxDepth)

4.5 Testing dell’algoritmo

La totalità dei casi di test svolti si trova nella classe **PestDepthCertifierTest**. In questa sezione illustreremo alcuni scenari significativi con *initDepth*=12, ovvero con 12 livelli che vanno da 12 a 1. La validità dei casi di test è stata inoltre certificata con l’uso del plugin per Eclipse *codecov* [8] che ha verificato la copertura dei casi di test in relazione al codice dell’algoritmo.

Una struttura di test in generale ha inizio con la creazione di una classe di tipo **PestDepthCertifier** e contiene più invocazioni del metodo **checkGoTo** che ha questa struttura:

Listing 4.3. Metodo checkGoTo

```
1 static void checkGoTo(DepthCertBase dp, DepthCertBase.Solution sol, int
2     nextDepth, Result result) {
3     dp.computeNextDepth(sol);
4     assertEquals(nextDepth, dp.getCurrentDepth());
5     assertEquals(result, dp.getCurrentStatus().currentResult);
6 }
```

Il metodo **checkGoTo** riceve come parametri: una classe di tipo **PestDepthCertifier** che estende **DepthCertBase**, la soluzione alla prova data dal paziente, il *nextDepth* che è il livello successivo atteso e *result* che è lo stato atteso della variabile *certifierStatus.currentResult*. Con questi parametri in ingresso viene invocato il metodo **computeNextDepth** che calcola il livello successivo (da confrontare con quello che ci attendiamo presente in *nextDepth*). In questo modo si riesce a simulare il funzionamento dell’algoritmo in tutte le sue possibili casistiche.

Ci siamo serviti di strumenti di debug per visualizzare il comportamento delle variabili dell’algoritmo aiutando la comprensione del suo funzionamento.

1. Scenario di test con profondità iniziale negativa

Listing 4.4. Test 01

```

1 public void test01() {
2     try {
3         new PestDepthCertifier(-12);
4         fail("Should throw exception");
5     } catch (IllegalArgumentException e) {
6         // Exception expected
7     }
8 }
```

In questo scenario particolare tentiamo di creare una classe con *initDepth* negativa. Non avrebbe senso avere una disparità iniziale negativa e quindi viene lanciata un’eccezione.

2. Scenario di test con risposte tutte corrette

Listing 4.5. Test 02

```

1 public void test02() {
2     PestDepthCertifier dp = new PestDepthCertifier(12);
3     checkGoTo(dp, Solution.NULL, 12, CONTINUE); // NULL (SKIP BUTTON)
4     checkGoTo(dp, Solution.RIGHT, 6, CONTINUE); // OK 12
5     checkGoTo(dp, Solution.RIGHT, 3, CONTINUE); // OK 6
6     checkGoTo(dp, Solution.RIGHT, 2, CONTINUE); // OK 3
7     checkGoTo(dp, Solution.RIGHT, 1, CONTINUE); // OK 2
8
9     checkGoTo(dp, Solution.RIGHT, 1, CONTINUE); // OK 1
10    checkGoTo(dp, Solution.RIGHT, 1, FINISH_CERTIFIED); // OK 1,
11        FINISH_CERTIFIED
11 }
```

In questo scenario testiamo il caso in cui il paziente risponda correttamente a tutti i livelli. Al livello iniziale viene testato inoltre lo *skip button*, il quale provoca un cambiamento dell’immagine mantenendo la stessa disparità. La risoluzione esatta del livello 12 mantiene intatto l’insieme e pone il prossimo livello da risolvere il livello 6. Risolvendolo correttamente, l’insieme viene ridotto a [6, 1]. Il livello 3 è il successivo che superato restringe l’insieme a [3, 1] e pone il livello 2 da superare. La risposta corretta permette di restringere

l’insieme a due livelli consecutivi [2, 1] la cui differenza è uguale a 1 causando il passaggio alla seconda parte dell’algoritmo. Da notare che nella prima parte, avendo risposto solo in maniera corretta, il calcolo del livello successivo viene sempre svolto calcolando la media degli estremi dell’insieme arrotondata per difetto. La seconda parte dell’algoritmo inizia testando l’estremo destro, ossia il livello 1. Due risposte consecutive corrette senza errori permettono di aumentare a +2 il suo valore nell’array e di terminare l’algoritmo restituendo come certificato il livello 1.

Listing 4.6. Debug 02

```

1 [PART ONE] [0,0,0,0,0,0,0,0,0,0,0,1] [CURR:12 RIGHT NEXT:6 L:12 R:1]
2 [PART ONE] [0,0,0,0,0,1,0,0,0,0,0,1] [CURR:6 RIGHT NEXT:3 L:6 R:1]
3 [PART ONE] [0,0,1,0,0,1,0,0,0,0,0,1] [CURR:3 RIGHT NEXT:2 L:3 R:1]
4 [PART ONE] [0,1,1,0,0,1,0,0,0,0,0,1] [CURR:2 RIGHT NEXT:1 L:2 R:1]
5 [PART TWO] [1,1,1,0,0,1,0,0,0,0,0,1] [CURR:1 RIGHT NEXT:1 L:2 R:1]
6 [PART TWO] [2,1,1,0,0,1,0,0,0,0,0,1] [CURR:1 RIGHT NEXT:1 L:2 R:1]
```

3. Scenario di test con fine non certificata immediata

Listing 4.7. Test 03

```

1 public void test03() {
2     PestDepthCertifier dp = new PestDepthCertifier(12);
3     checkGoTo(dp, Solution.WRONC, 12, CONTINUE); // NO 12 (chance = 0)
4     checkGoTo(dp, Solution.WRONC, 12, FINISH_NOT_CERTIFIED); // NO 12,
    FINISH_NOT_CERTIFIED
5 }
```

In questo scenario è mostrato il caso in cui un doppio errore al livello iniziale provoca la terminazione dell’algoritmo con un’uscita non certificata. Il paziente non è riuscito a certificare di saper svolgere alcun livello avendo sbagliato due volte consecutive il livello più facile. La variabile *chance* tiene traccia dei tentativi sul livello massimo e in caso di errore viene decrementata di 1. Essa è inizializzata a 1 e un secondo errore provoca un’uscita non certificata.

Listing 4.8. Debug 03

```

1 [PART ONE] [0,0,0,0,0,0,0,0,0,0,0,0] [CURR:12 WRONG NEXT:0 L:12 R:1]
2 [PART ONE] [0,0,0,0,0,0,0,0,0,0,0,0] [CURR:12 WRONG NEXT:0 L:12 R:1]
```

4. Scenario di test con doppio errore consecutivo non al livello iniziale

Listing 4.9. Test 04

```

1 public void test04() {
2     PestDepthCertifier dp = new PestDepthCertifier(12);
3     checkGoTo(dp, Solution.RIGHT, 6, CONTINUE); // OK 12
4     checkGoTo(dp, Solution.RIGHT, 3, CONTINUE); // OK 6
5     checkGoTo(dp, Solution.RIGHT, 2, CONTINUE); // OK 3
6     checkGoTo(dp, Solution.WRONG, 2, CONTINUE); // NO 2
7
8     checkGoTo(dp, Solution.WRONG, 3, CONTINUE); // NO 2 (HEAVY -> -4), SHIFT
9         TO 3
10    checkGoTo(dp, Solution.RIGHT, 3, FINISH_CERTIFIED); // OK 3,
11        FINISH_CERTIFIED
12 }
```

In questo scenario testiamo il caso avente un doppio errore consecutivo non iniziale. Il paziente come nel primo test risolve correttamente il livello 12 (un errore sul primo livello è concesso grazie alla variabile *chance*) e in successione risolve anche i livelli 6 e 3. L’insieme è ristretto a [3, 1] e il livello 2 è il prossimo da testare. Un errore al livello 2 provoca la restrizione dell’insieme a [3, 2] e il conseguente passaggio alla seconda parte dell’algoritmo. In questa parte viene testato il livello 2 essendo l’estremo destro (*rightLimit*) e un errore provoca il cosiddetto shift al livello superiore. Il livello sottoposto è il 3 e una risposta giusta aumenta il valore dell’array di +1 raggiungendo il valore +2. L’algoritmo termina e restituisce la certificazione del livello 3.

Listing 4.10. Debug 04

```

1 [PART ONE] [0,0,0,0,0,0,0,0,0,0,0,1] [CURR:12 RIGHT NEXT:6 L:12 R:1]
2 [PART ONE] [0,0,0,0,0,1,0,0,0,0,0,1] [CURR:6 RIGHT NEXT:3 L:6 R:1]
3 [PART ONE] [0,0,1,0,0,1,0,0,0,0,0,1] [CURR:3 RIGHT NEXT:2 L:3 R:1]
4 [PART ONE] [0,-1,1,0,0,1,0,0,0,0,0,1] [CURR:2 WRONG NEXT:2 L:3 R:2]
5 [PART TWO] [0,-4,1,0,0,1,0,0,0,0,0,1] [CURR:2 WRONG NEXT:3 L:3 R:2]
6 [PART TWO] [0,-4,2,0,0,1,0,0,0,0,0,1] [CURR:3 RIGHT NEXT:3 L:3 R:2]
```

5. Scenario di test con molti errori consecutivi che provocano un’uscita non certificata a causa del fallimento al livello massimo

Listing 4.11. Test 05

```

1 public void test05() {
2     PestDepthCertifier dp = new PestDepthCertifier(12);
3     checkGoTo(dp, Solution.RIGHT, 6, CONTINUE); // OK 12
4     checkGoTo(dp, Solution.WRONG, 9, CONTINUE); // NO 6
5     checkGoTo(dp, Solution.WRONG, 11, CONTINUE); // NO 9
6     checkGoTo(dp, Solution.WRONG, 11, CONTINUE); // NO 11
7
8     checkGoTo(dp, Solution.WRONG, 12, CONTINUE); // NO 11 (HEAVY -> -4),
9         SHIFT TO 12
10    checkGoTo(dp, Solution.WRONG, 12, CONTINUE); // NO 12
11    checkGoTo(dp, Solution.WRONG, 12, FINISH_NOT_CERTIFIED); // NO 12 (HEAVY
-> -3) AND MAXDEPTH REACHED, FINISH_NOT_CERTIFIED
11 }
```

In questo scenario il paziente nella prima parte risponde correttamente al livello 12 e sbaglia consecutivamente i livelli 6, 9 e 11 restringendo l’insieme a [12, 11]. Nella seconda parte dell’algoritmo viene subito testato il livello 11 essendo il l’estremo destro; un errore su esso provoca un valore di -4 nell’array (-1 per averlo sbagliato nella prima parte e -3 per il secondo errore nella seconda parte con $weight = 3$). Quindi si sottopone il livello 12 che era stato risolto correttamente nella prima parte (valore +1 nell’array). Due errori su questo livello provocano rispettivamente una sottrazione di -1 e un -3 (peso applicato) nell’array. Andando al di sotto del valore -2 nell’array anche per il livello 12 (il livello massimo nonchè quello iniziale), l’algoritmo restituisce una fine non certificata.

Listing 4.12. Debug 05

```

1 [PART ONE] [0,0,0,0,0,0,0,0,0,0,0,1] [CURR:12 RIGHT NEXT:6 L:12 R:1]
2 [PART ONE] [0,0,0,0,0,-1,0,0,0,0,0,1] [CURR:6 WRONG NEXT:9 L:12 R:6]
3 [PART ONE] [0,0,0,0,0,-1,0,0,-1,0,0,1] [CURR:9 WRONG NEXT:11 L:12 R:9]
4 [PART ONE] [0,0,0,0,0,-1,0,0,-1,0,-1,1] [CURR:11 WRONG NEXT:11 L:12 R:11]
5 [PART TWO] [0,0,0,0,0,-1,0,0,-1,0,-4,1] [CURR:11 WRONG NEXT:12 L:12 R:11]
6 [PART TWO] [0,0,0,0,0,-1,0,0,-1,0,-4,0] [CURR:12 WRONG NEXT:12 L:12 R:11]
7 [PART TWO] [0,0,0,0,0,-1,0,0,-1,0,-4,-3] [CURR:12 WRONG NEXT:12 L:12 R:11]
```

6. Scenario di test con esempio di shift

Listing 4.13. Test 06

```

1 public void test06() {
2     PestDepthCertifier dp = new PestDepthCertifier(12);
3     checkGoTo(dp, Solution.RIGHT, 6, CONTINUE); // OK 12
4     checkGoTo(dp, Solution.RIGHT, 3, CONTINUE); // OK 6
5     checkGoTo(dp, Solution.WRONG, 5, CONTINUE); // NO 3
6     checkGoTo(dp, Solution.RIGHT, 4, CONTINUE); // OK 5
7     checkGoTo(dp, Solution.WRONG, 4, CONTINUE); // NO 4
8
9     checkGoTo(dp, Solution.WRONG, 5, CONTINUE); // NO 4 (HEAVY -> -4), SHIFT
10    TO 5
11    checkGoTo(dp, Solution.RIGHT, 5, FINISH_CERTIFIED); // OK 5,
12        FINISH_CERTIFIED
13 }
```

In questo scenario nella prima parte dell'algoritmo il paziente risponde correttamente al livello 12 e in successione anche al livello 6 riducendo l'insieme a [6, 1]. Il livello sottoposto è il 3 che viene sbagliato e porta ad avere un insieme [6, 3]. Si testa il livello 5 che viene risolto correttamente riducendo l'insieme a [5, 3]. Un errore al livello 4 causa una riduzione dell'insieme a [5, 4]. Nella seconda parte dell'algoritmo viene testato per primo il livello 4 che, sbagliato dal paziente, causa a uno shift al livello superiore (l'errore aveva peso 3 e causa il superamento della banda di uscita -2). Il livello 5 viene risolto in maniera corretta e, aggiungendo +1 nell'array, il valore contenuto raggiunge +2 e quindi viene restituito come livello certificato il livello 5.

Listing 4.14. Debug 06

```

1 [PART ONE] [0,0,0,0,0,0,0,0,0,0,0,1] [CURR:12 RIGHT NEXT:6 L:12 R:1]
2 [PART ONE] [0,0,0,0,0,1,0,0,0,0,0,1] [CURR:6 RIGHT NEXT:3 L:6 R:1]
3 [PART ONE] [0,0,-1,0,0,1,0,0,0,0,0,1] [CURR:3 WRONG NEXT:5 L:6 R:3]
4 [PART ONE] [0,0,-1,0,1,1,0,0,0,0,0,1] [CURR:5 RIGHT NEXT:4 L:5 R:3]
5 [PART ONE] [0,0,-1,-1,1,1,0,0,0,0,0,1] [CURR:4 WRONG NEXT:4 L:5 R:4]
6 [PART TWO] [0,0,-1,-4,1,1,0,0,0,0,0,1] [CURR:4 WRONG NEXT:5 L:5 R:4]
7 [PART TWO] [0,0,-1,-4,2,1,0,0,0,0,0,1] [CURR:5 RIGHT NEXT:5 L:5 R:4]
```

7. Scenario di test con molti errori fino alla certificazione del livello massimo

Listing 4.15. Test 07

```
1 public void test07() {
2     PestDepthCertifier dp = new PestDepthCertifier(12);
3     checkGoTo(dp, Solution.RIGHT, 6, CONTINUE); // OK 12
4     checkGoTo(dp, Solution.WRONG, 9, CONTINUE); // NO 6
5     checkGoTo(dp, Solution.RIGHT, 7, CONTINUE); // OK 9
6     checkGoTo(dp, Solution.WRONG, 8, CONTINUE); // NO 7
7     checkGoTo(dp, Solution.WRONG, 8, CONTINUE); // NO 8
8
9     checkGoTo(dp, Solution.WRONG, 9, CONTINUE); // NO 8 (HEAVY -> -4), SHIFT
10    TO 9
11    checkGoTo(dp, Solution.WRONG, 9, CONTINUE); // NO 9
12    checkGoTo(dp, Solution.WRONG, 10, CONTINUE); // NO 9 (HEAVY -> -3),
13    SHIFT TO 10
14    checkGoTo(dp, Solution.WRONG, 10, CONTINUE); // NO 10
15    checkGoTo(dp, Solution.WRONG, 11, CONTINUE); // NO 10 (HEAVY -> -4),
16    SHIFT TO 11
17    checkGoTo(dp, Solution.WRONG, 11, CONTINUE); // NO 11
18    checkGoTo(dp, Solution.WRONG, 12, CONTINUE); // NO 11 (HEAVY -> -4),
19    SHIFT TO 12
20    checkGoTo(dp, Solution.RIGHT, 12, FINISH_CERTIFIED); // OK 12,
21    FINISH_CERTIFIED
22 }
```

In questo scenario mostreremo come tanti errori consecutivi nella seconda parte dell’algoritmo provochino un susseguirsi a catena di livelli sempre più facili sottoposti al paziente. Brevemente, nella prima parte viene raggiunto l’insieme [9, 8] a seguito di errori sui livelli 6 e 8. La seconda parte ha inizio con il test sul livello 8 ed un suo errore provoca lo shift al livello 9; infatti il livello 8 era stato già sbagliato una volta nella prima parte dell’algoritmo. Ora si presentano in successione livelli sempre più facili e immediatamente superiori al precedentemente sbagliato. Si sottopone il livello 9 che con due errori viene scartato. Nonostante il livello 9 sia stato risolto correttamente nella prima parte, il secondo errore ha un peso triplo e provoca inevitabilmente l’uscita. In seguito si prosegue con i livelli 10 e poi 11 a cui il paziente risponde con un

doppio errore. Si passa al livello 12 che ha valore +1 nell'array. Il livello viene superato dal paziente e provoca un'uscita certificata restituendo i livello 12.

Listing 4.16. Debug 07

```
1 [PART ONE] [0,0,0,0,0,0,0,0,0,0,0,0,1] [CURR:12 RIGHT NEXT:6 L:12 R:1]
2 [PART ONE] [0,0,0,0,0,-1,0,0,0,0,0,1] [CURR:6 WRONG NEXT:9 L:12 R:6]
3 [PART ONE] [0,0,0,0,0,-1,0,0,1,0,0,1] [CURR:9 RIGHT NEXT:7 L:9 R:6]
4 [PART ONE] [0,0,0,0,0,-1,-1,0,1,0,0,1] [CURR:7 WRONG NEXT:8 L:9 R:7]
5 [PART ONE] [0,0,0,0,0,-1,-1,-1,1,0,0,1] [CURR:8 WRONG NEXT:8 L:9 R:8]
6 [PART TWO] [0,0,0,0,0,-1,-1,-4,1,0,0,1] [CURR:8 WRONG NEXT:9 L:9 R:8]
7 [PART TWO] [0,0,0,0,0,-1,-1,-4,0,0,0,1] [CURR:9 WRONG NEXT:9 L:9 R:8]
8 [PART TWO] [0,0,0,0,0,-1,-1,-4,-3,0,0,1] [CURR:9 WRONG NEXT:10 L:9 R:8]
9 [PART TWO] [0,0,0,0,0,-1,-1,-4,-3,-1,0,1] [CURR:10 WRONG NEXT:10 L:9 R:8]
10 [PART TWO] [0,0,0,0,0,-1,-1,-4,-3,-4,0,1] [CURR:10 WRONG NEXT:11 L:9 R:8]
11 [PART TWO] [0,0,0,0,0,-1,-1,-4,-3,-4,-1,1] [CURR:11 WRONG NEXT:11 L:9 R:8]
12 [PART TWO] [0,0,0,0,0,-1,-1,-4,-3,-4,-4,1] [CURR:11 WRONG NEXT:12 L:9 R:8]
13 [PART TWO] [0,0,0,0,0,-1,-1,-4,-3,-4,-4,2] [CURR:12 RIGHT NEXT:12 L:9 R:8]
```

Appendice A

AsyncTask

Il pattern di un AsyncTask usato dall'applicazione per la connessione con la servlet (codice proveniente da *MainPatient.java*).

Listing A.1. Porzione di codice di AsyncTask

```
1 new AsyncTask<Void, Void, Void>() {
2
3     public Void doInBackground(Void... params) {
4         // URI BUILDER
5         Uri.Builder builder = new Uri.Builder();
6         builder.scheme("https")
7             .encodedAuthority(DefaultValues.AUTHORITY)
8             .appendPath("se4medservice")
9             .appendPath("")
10            .appendQueryParameter(DefaultValues.ACTION_PARAM_NAME,
11                DefaultValues.DELETEPATIENTDOC_ACTION)
12            .appendQueryParameter(DefaultValues.EMAIL_PARAM, doctorMail)
13            .appendQueryParameter(DefaultValues.PASSWORD_PARAM,
14                doctorPassword)
15            .appendQueryParameter(DefaultValues.IDPATIENT_PARAM,
16                String.valueOf(idPatient))
17            .appendQueryParameter(DefaultValues.IDAPP_PARAM,
18                DefaultValues.STEREOTEST); // Insert semicolon
19
20        String myUri = builder.build().toString();
21        System.out.println("URI: " + myUri);
22    }
23}
```

```
19     // URL CONNECTION
20
21     HttpURLConnection urlConnection = null;
22     StringBuilder result = new StringBuilder();
23
24     try {
25         URL url = new URL(myUri);
26         urlConnection = (HttpURLConnection) url.openConnection();
27         urlConnection.setRequestMethod("GET");
28         urlConnection.setReadTimeout(10000 /* milliseconds */);
29         urlConnection.setConnectTimeout(15000 /* milliseconds */);
30         urlConnection.setDoOutput(true);
31         urlConnection.connect();
32
33         int responseCode = urlConnection.getResponseCode();
34         System.out.println("RESPONSE CODE: " + responseCode);
35
36         if ((responseCode == HttpURLConnection.HTTP_OK) || (responseCode ==
37             HttpURLConnection.HTTP_CREATED)) {
38             InputStream in = new
39                 BufferedInputStream(urlConnection.getInputStream());
40             BufferedReader reader = new BufferedReader(new
41                 InputStreamReader(in));
42             String line;
43             while ((line = reader.readLine()) != null) {
44                 result.append(line);
45             }
46         } else {
47             // Toast (Handler)
48             Handler handler = new Handler(Looper.getMainLooper());
49             handler.post(new Runnable() {
50                 public void run() {
51                     Toast t = Toast.makeText(getApplicationContext(),
52                         "Network error (HTTP)", Toast.LENGTH_LONG);
53                     t.show();
54                 }
55             });
56         }
57     } catch (Exception e) {
58         // Toast (Handler)
59         Handler handler = new Handler(Looper.getMainLooper());
```

```
55         handler.post(new Runnable() {
56             public void run() {
57                 Toast t = Toast.makeText(getApplicationContext(), "Network
58                     error (E)", Toast.LENGTH_LONG);
59                     t.show();
60             }
61         );
62         e.printStackTrace();
63     } finally {
64         if (urlConnection != null) {
65             urlConnection.disconnect();
66         }
67     }
68
69     try {
70         String response = "[" +
71             result.toString().substring(result.toString().indexOf("{") + 1,
72             result.toString().lastIndexOf("}") + "}]";
73         System.out.println("RESPONSE: " + response);
74
75         JSONObject jsonobject = new JSONObject(response);
76         final String status = jsonobject.getString("status");
77         System.out.println("STATUS: " + status);
78
79         if (status.equals(DefaultValues.STATUS_OK)) {
80             SharedPreferences.Editor e = SPPatient.edit();
81             e.putString(DefaultValues.ACTUAL_PATIENT_NAME, "404");
82             e.putString(DefaultValues.ACTUAL_PATIENT_SURNAME, "404");
83             e.putInt(DefaultValues.ACTUAL_PATIENT_ID, -1);
84             e.apply();
85
86             // Toast (Handler)
87             Handler handler = new Handler(Looper.getMainLooper());
88             handler.post(new Runnable() {
89                 public void run() {
90                     Toast t = Toast.makeText(getApplicationContext(),
91                         "Patient deleted", Toast.LENGTH_LONG);
92                     t.show();
93                 }
94             );
95         }
96     }
97 }
```

```
91     } else {
92         // Toast (Handler)
93         Handler handler = new Handler(Looper.getMainLooper());
94         handler.post(new Runnable() {
95             public void run() {
96                 Toast t = Toast.makeText(getApplicationContext(),
97                     "Status: " + status, Toast.LENGTH_LONG);
98                 t.show();
99             }
100        });
101    } catch (Exception e) {
102        e.printStackTrace();
103    }
104    return null;
105 }
```

Bibliografia

- [Car17] Neil Carter. *PEST (Parameter Estimation by Sequential Testing)*. 2017.
URL: psy.swansea.ac.uk/staff/carter/projects/PEST_explanation.htm.
- [Hum17] Humanitas. *Ambliopia*. 2017. URL: www.humanitas.it/malattie/ambliopia-occhio-pigro.
- [Pen80] Alex Pentland. *Maximum likelihood estimation: The best PEST*. 1980.
- [TC67] M. M. Taylor e C. Douglas Creelman. *PEST: Efficient Estimates on Probability Functions*. 1967.

Sitografia

- [1] *3D4AMB*. URL: 3d4amb.unibg.it.
- [2] *SE4MED*. URL: se4med.unibg.it.
- [3] *Android Developers*. URL: developer.android.com.
- [4] *Eclipse*. URL: eclipse.org.
- [5] *MySQL*. URL: mysql.com.
- [6] *JSON*. URL: json.org.
- [7] *JOOQ*. URL: jooq.org.
- [8] *CodeCover*. URL: codecover.org.