

**Progetto Giocoso 2015**

Documentazione delle Applicazioni

*A cura di Lorenzo Affetti*

(lorenzo.affetti@polimi.it)

*Ultimo aggiornamento: 27 ottobre 2015*

**Indice**

**1 Chef per un Giorno 2**

**2 Lego Digital Sonoro 1 18**

**3 Lego Digital Sonoro 2 34**

**4 Trova l’Intruso 1 51**

**5 Trova l’Intruso 2 64**

**6 SuperApp 76**

**Capitolo 1**

**Chef per un Giorno**

*Documentazione a cura di Giovanni Quattrocchi* (giovanni.quattrocchi@polimi.it)

**Indice**

1.1 Manuale Sviluppatore . . . . . . . . . . . . . . . . . . . . . . . 4

1.1.1 Diagramma delle classi del modello . . . . . . . . . . . 4

1.2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1.1.2 | AllJoyn . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 4 |
| 1.1.3 | Gimbal . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 5 |
| 1.1.4 | Activities | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 6 |
| 1.1.5 | Adapters . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 7 |
| Manuale Utente . . . . . . . . . . . . . . . . . . . . . . . . . . 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1.2.1 | Introduzione . | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 8 |
| 1.2.2 | Preparazione | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 9 |
| 1.2.3 | Elenco Ingredienti . | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 11 |
| 1.2.4 | Il gioco . . . . . . . | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 12 |

**1.1 Manuale Sviluppatore**

**1.1.1 Diagramma delle classi del modello**

Le classi del modello sono le seguenti:

**Game** classe statica che permette il caricamento dei dati di gioca da JSON

e mantiene una referenza ai livelli e ai piatti

**Level** classe astratta, il metodo getNumberOfIngredientForDish ritorna il numero corretto di elementi solo per il Livello 1

**Dish** il piatto, un insieme di ingredienti

**Ingredient** l’ingrediente

**Turn** tiene traccia del completamento di un piatto, invia callback al relativo

TurnListener. Al termine del piatto genera un TurnResult. **TurnListener** interfaccia per la ricezioni di eventi relativi a un turno **TurnResult** il risultato del turno

**1.1.2 AllJoyn**

La modalit`a multiplayer `e stata realizzata attraverso AllJoyn. Qui si pu`o tro-

vare la documentazione completa: [https://allseenalliance.org/dev](https://allseenalliance.org/developers/develop/api-guide/core/android)elopers/ [develop/api-guide/core/android](https://allseenalliance.org/developers/develop/api-guide/core/android). L’applicazione espone un generico Mul- tiPlayerService con il metodo sendMessage(). Questo metodo permette di inviare semplici Signal con un parametro stringa. Il formato di invio dei messaggi `e CSV, il primo elemento `e il tipo del messaggio. Verranno inviati

tre tipi di messaggi:

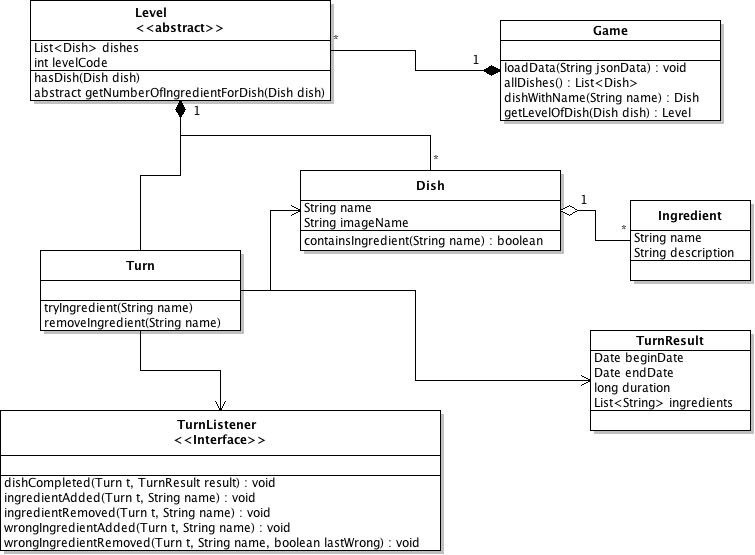


Figura 1.1: Class Diagram

1. dishes seguito dai nomi del piatti, rappresenta l’inizio del gioco, l’host della partita invia al joiner della sessione i piatti selezionati

2. stop inviato quando uno dei due dispositivi abbondona il gioco (alla ri- cezione di questo messaggio si torna indietro alla schermata principale)

3. dish seguito dal nome del piatto, indica il completamento di un piatto

**1.1.3 Gimbal**

Nel gioco ciascun ingrediente `e un trasmettitore Gimbal. Nell’applicazione si fa uso della versione 2 dell’API, in particolare si fa uso della parte di Beacon management. Qui la documentazione: [https://gimbal.com/doc/android/](https://gimbal.com/doc/android/v2/devguide.html#set_beacon)

[v2/devguide.html#set\_beacon](https://gimbal.com/doc/android/v2/devguide.html#set_beacon) Tale API viene utilizzata nella MainActivi- ty per orchestrare il flow del gioco. I beacon sono associati ad un ingrediente attraverso il name. La vicinanza di un beacon `e determinata dal valore del suo RSSI. Il valore che discrimina la vicinanza di un beacon `e -60. Atten- zione: potrebbe essere necessario cambiare il threshold di vicinanza (si trova nella classe Commons.java).

**1.1.4 Activities**

L’applicazione dispone di quattro Activity:

**StartActivity** mostra la schermata principale e carica i dati del gioco per evitare successivi caricamenti. Dispone di un bottone per l’inserimento della email per l’invio delle statistiche del gioco. La persistenza della email `e affidata alle SharedPreferences di Android

**ChooseSessionActivity** mostra la schermata di scelta dei piatti, cerca nell’Intent un Extra “mode” che indica la modalit`a di gioco (singo- la o multiplayer). La selezione dei piatti pu`o avvenire in due modi: selezionandoli da una GridView o premendo il tasto “Casuale”

**MultiPlayerConfigActivity** gestisce parte dall’inizializzazione della logi- ca di AllJoyn, il comportamento varia a seconda se il giocatore seleziona di creare una partita (ruolo host) o di accedere ad una partita (ruolo joiner)

**MainActivity** orchestra la partita usando Gimbal API, AllJoyn Framework, la classe Turn e TurnListener. Cerca nell’Intent un Extra “mode” per la modalit`a e uno “dishes” per i piatti.

Di seguito viene riportato il flow di accesso alle diverse activity.

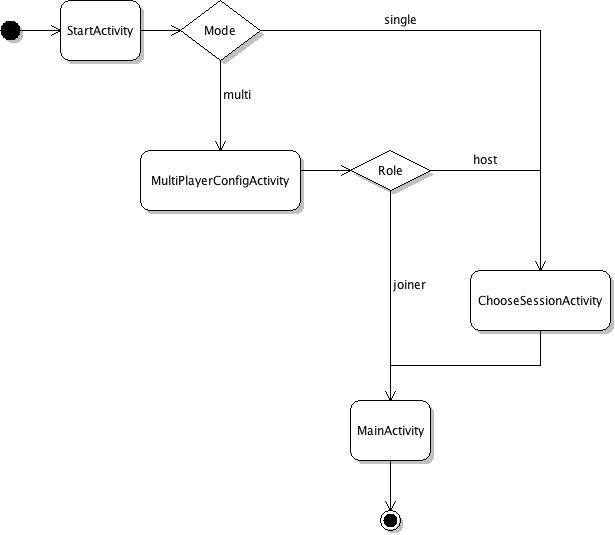


Figura 1.2: Flow di accesso alle activity

Durante una partita il tasto Back di Android riporta alla schermata inizia- le dopo una conferma. Nel caso di Multiplayer se un dispositivo abbandona la partita anche l’altro torna alla schermata principale.

**1.1.5 Adapters**

Per visualizzare liste di oggetti sono stati usati i seguenti Adapter:

**DishAdapter** viene utilizzato nella schermata di selezione dei piatti, tiene traccia degli elementi selezionati

**IngredientAdapter** controlla la visualizzazione delle interazioni tra ingre- dienti e dispositivo. Mostra immagini diverse a seconda dell’assenza di ingredienti, ingredienti presenti o errati

**TableAdapter** controlla la visualizzazione della tavola con i sottopiatti vuo- ti o riempiti da quelli completati dalla squadra relativa al dispositivo e anche dalla squadra avversaria in caso di multiplayer.

Altre classi e file

**Commons** contiene costanti, della logica condivisa durante il gioco (Al- lJoyn) e metodi statici (invio mail)

**GMailSender** e JSSEProvider classi helper per l’invio di email

**PrefUtils** classe helper per la gestione delle SharedPreferences

/raw/data.json file contente i dati del gioco (ingredienti, livelli, piatti)

/raw/data\_source.csv file originale in csv in cui erano contenuti i dati del gioco non ben strutturati

/raw/gen\_json.py script python per trasformare data\_source.csv in data. json

**1.2 Manuale Utente**

**1.2.1 Introduzione**

Chef per Un Giorno `e un gioco per dispositivi Android. Lo scopo del gioco `e indovinare gli ingredienti necessari per la preparazione di un piatto attraverso

l’avvicinamento di trasmettitori Gimbal al dispositivo. Sono supportate due modalit`a: modalit`a singola (una squadra) e modalit`a multipla (due squadre).

**ATTENZIONE**: per il corretto funzionamento dell’applicazione `e ne- cessario attivare sia il Wi-Fi sia il Bluetooth.

**1.2.2 Preparazione**

Per giocare `e necessario configurare i trasmettitori Gimbal. Le credenziali d’accesso al portale Gimbal ([https://manager.gimbal.com](https://manager.gimbal.com/)) verrano fornite su richiesta scrivendo una email a [giovanni.quattrocchi@polimi.it.](mailto:giovanni.quattrocchi@polimi.it)

1. Entrare nel portale Gimbal all’indirizzo [https://manager.gimbal. com](https://manager.gimbal.com/)

2. Selezionare nella barra laterale Beacons (vedi figura 1.3)

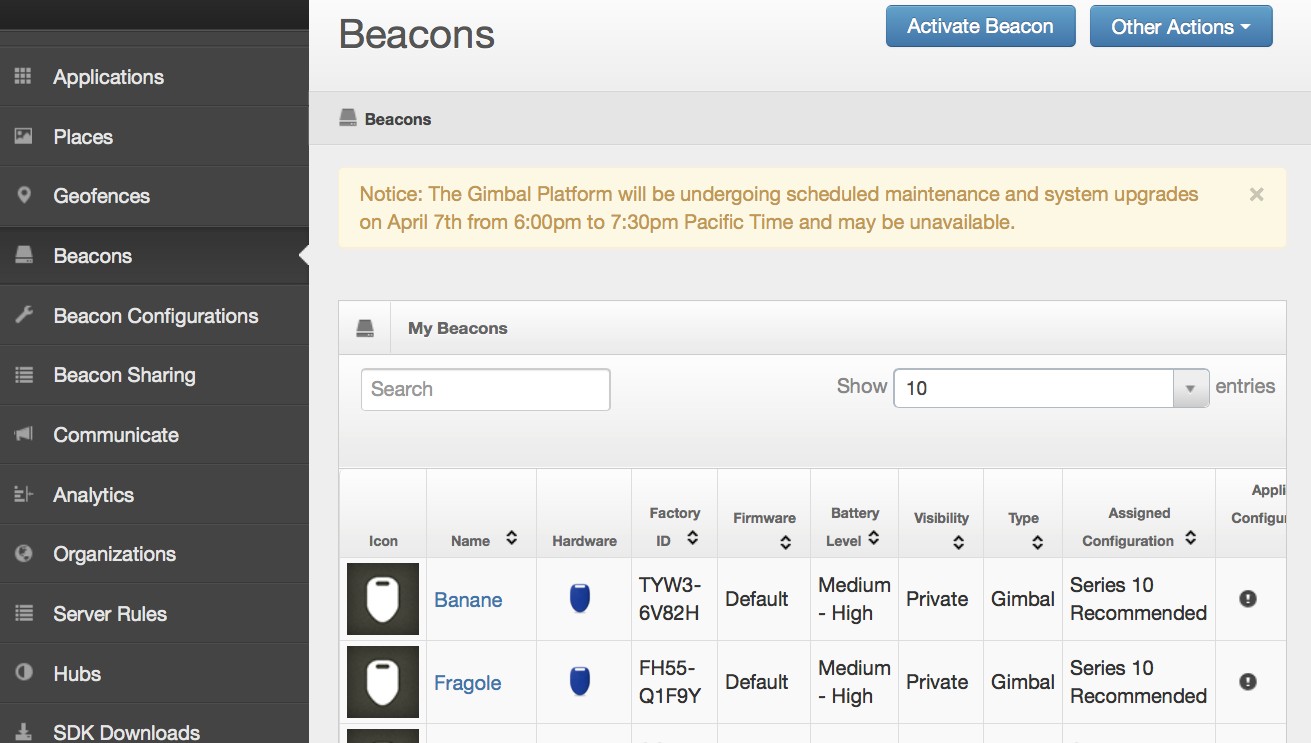


Figura 1.3: La schermata contenente il listato dei beacon nel portale Gimbal

3. Selezionare il bottone Activate Beacon

4. Aprire un trasmettitore Gimbal e trovare il Factory ID (per come trovare il Factory ID, vedi figura 1.4)

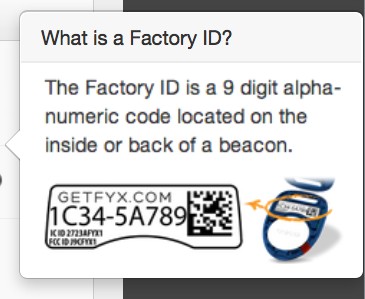


Figura 1.4: Il factory ID di un beacon

5. Inserire come nome del Beacon l’ingrediente esattamente come riporta- to nell’ElencoIngredienti (fornito di seguito) e il Factory ID (vedi figura

1.5)

6. Confermare con Activate Beacon

7. Ripetere per tutti i trasmettiri/ingredienti

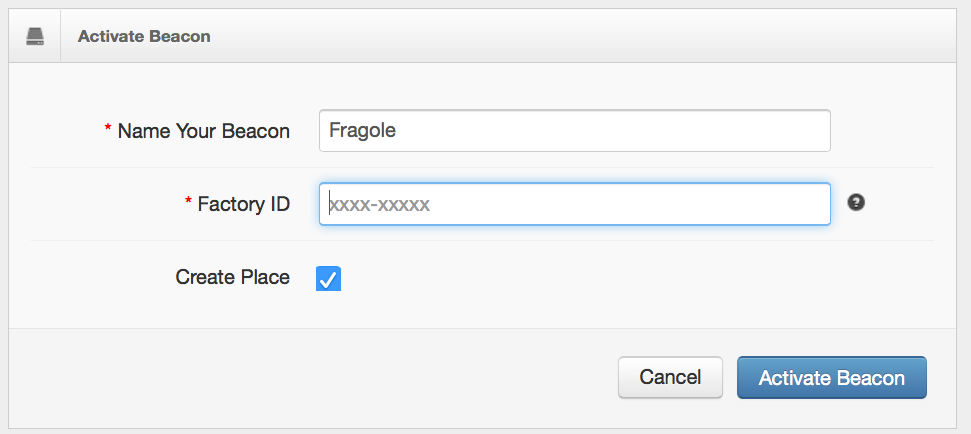


Figura 1.5: Schermata di attivazione di un beacon nel portale Gimbal

**1.2.3 Elenco Ingredienti**

*•* Ciliegie

*•* Mirtilli

*•* Uova

*•* Pomodori

*•* Insalata verde

*•* Piselli

*•* Zucchero

*•* Zucchine

*•* Banane

*•* Gamberi

*•* Mele

*•* Kiwi

*•* Riso

*•* Fagioli

*•* Farina

*•* Latte

*•* Peperoni

*•* Pasta

*•* Mozzarella

*•* Fragole

*•* Mango

*•* Carote

*•* Pollo

*•* Ceci

*•* Cetrioli

*•* Cipolle

**1.2.4 Il gioco**

**Schermata Iniziale e principi generali**

La prima schermata (vedi figura 1.6) permette di scegliere tra la modalit`a singola o multipla. In basso al centro della schermata `e anche possibile in- serire un indirizzo mail per l’invio dei dati di gioco. Per tornare indietro



Figura 1.6: Lo splash screen di Chef per un Giorno

a questa schermata dalla altre `e necessario selezionare il pulsante indietro nella barra inferiore (indicato con un triangolo orientato verso sinistra). Il tavolo di gioco comprende il dispositivo Android e uno spazio dove appog- giare i trasmettitori. Prima di preparare qualsiasi piatto allontanare tutti i trasmettitori dal tavolo di gioco.

**Modalit`a Singola**

Se si seleziona modalit`a singola comparir`a la schermata di selezione dei piat-

ti (vedi figura 1.7).

E` possibile scegliere due o quattro piatti da preparare.

Dopo averli selezionati cliccare il tasto Ok per continuare. In alternativa `e possibile generale una partita con piatti casuali (`e casuale anche il numero di

piatti) schiacciando il tasto Casuale. Dopo la scelta dei piatti comparir`a la schermata di gioco principale (vedi figura 1.8). A sinistra in alto comparir`a il piatto da preparare, sotto di questo, se il piatto appartiene al Livello 1, appariranno anche tante pentole arancioni quanti sono il numero degli ingre- dienti. Se il piatto appartiene al Livello 2 non comparir`a nulla. Avvicinando un trasmettitore al tavolo di gioco comparir`a una pentola verde se l’ingre- diente appartiene al piatto, rossa se non appartiene. La pentola rossa indica che c’`e almeno un ingrediente errato (possono essere anche due, tre, e cos`ı via) e scomparir`a quando tutti gli ingredienti errati saranno allontanati dal tavolo di gioco. Un piatto `e completato quando vengono avvicinati tutti gli ingredienti corretti e nessuno errato. Una volta completato il piatto, que- sto verr`a aggiunto alla tavola e verr`a mostrato il nuovo piatto da preparare. Quando tutti i piatti sono preparati il gioco `e terminato. Per tornare alla schermata principale cliccare il tasto indietro di Android.

**Modalit`a Multiplayer**

E` molto simile alla modalit`a singola con le seguenti varianti:

*•* I due dispositivi devono essere connessi alla stessa rete Wi-Fi

*•* I due tavoli di gioco devono essere lontani l’uno dall’altro per evitare interferenze

*•* Prima della selezione dei piatti `e necessario che su un dispositivo si selezioni Crea una partita e sull’altro Entra in una partita

*•* E` obbligatorio selezionare quattro piatti (2 per squadra)

*•* I quattro piatti verrano scelti dal dispositivo che crea la partita, l’altro rimarr`a in attesa fino a selezione avvenuta



Figura 1.7: La schermata di selezione dei piatti

*•* Giocano due squadre quella arancione e quella viola, il colore della squa- dra `e indicato nella schermata di gioco dal colore presente nell’angolo in basso a destra.

*•* Quando un piatto `e completato da una squadra, questo compare anche nella tavola della squadra avversaria

*•* Se su uno dei due dispositivi esce dal gioco, anche l’altro abbandoner`a il gioco

In figura 1.9 `e mostrata la schermata di gioco in modalit`a multiplayer.

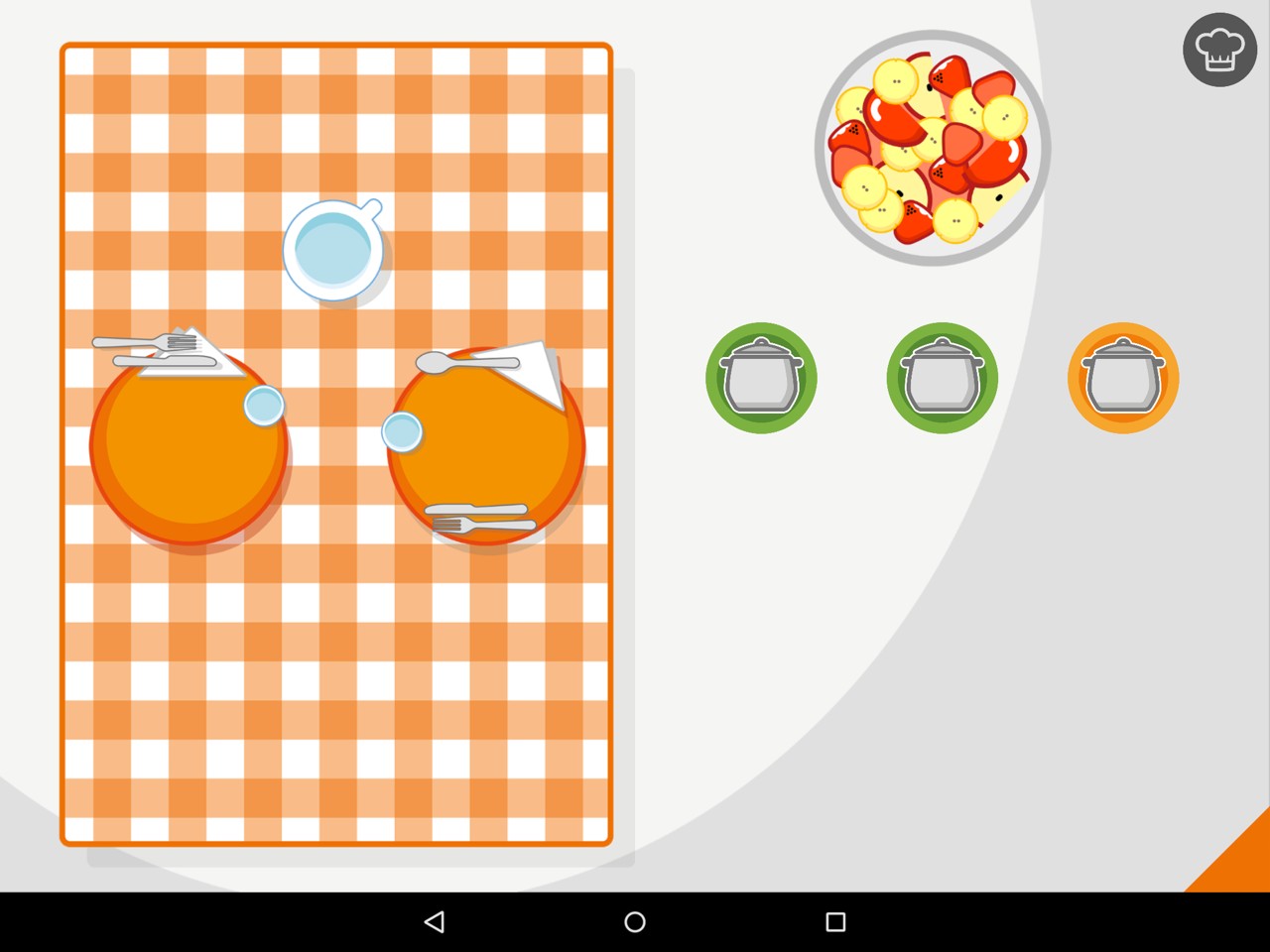


Figura 1.8: La schermata di gioco in modalit`a single player

**Invio dei dati di gioco**

Se `e stata inserita una mail valida (vedi Schermata Principale) verr`a inviata una mail con i seguenti dati per ogni piatto realizzato:

*<*nome del piatto*>*, *<*timestamp inizio*>*, *<*timestamp fine*>*, *<*durata in secondi*>*, *<*ingredienti in ordine di rilevazione*>*, *<*errori*>*

Es:

Macedonia 1, 1428877207567, 1428877307567, 100, Mele, Fragole, Bana- ne, errors(Carote, Pomodori)

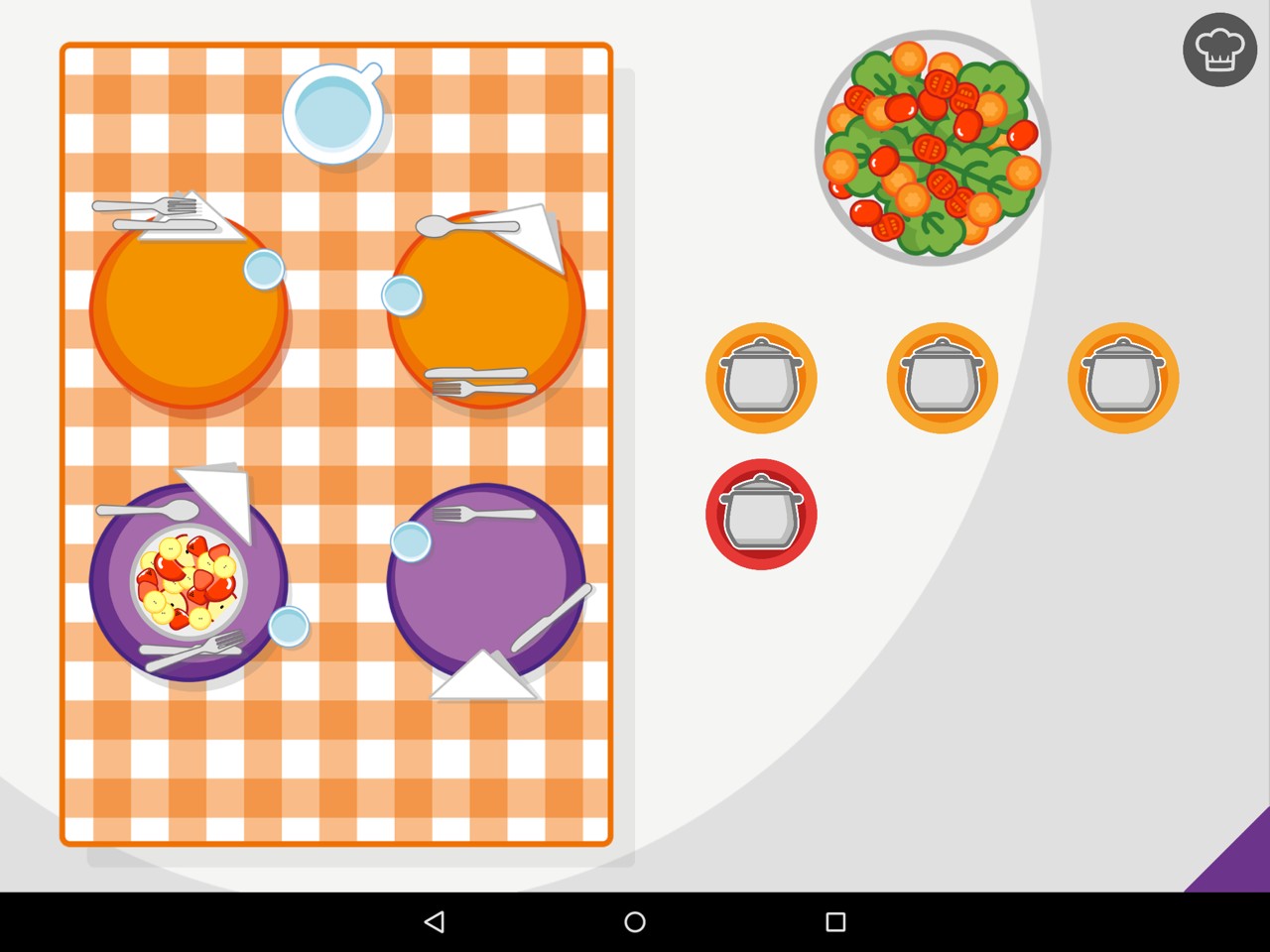


Figura 1.9: La schermata di gioco in modalit`a multiplayer

**Capitolo 2**

**Lego Digital Sonoro 1**

*Documentazione a cura di Chiara Calabrese* (chiara.calabrese@mail.polimi.it)

**Indice**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2.1 Manuale Utente . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 20 |
| 2.2 Manuale Sviluppatore . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 25 |
| 2.2.1 Partita singola | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 28 |

2.2.2 Partita multipla . . . . . . . . . . . . . . . . . . . . . . 31

**2.1 Manuale Utente**

L’applicazione LegoDigitalSonoro `e una applicazione mobile progettata per la compatibilit`a con Android. L’applicazione `e un semplice gioco dedicato ai bambini in et`a prescolare il cui intento `e quello di insegnare loro a comporre delle parole bisillabe.

Al giocatore vengono presentate 4 o 6 tessere rappresentanti una sillaba di una determinata parola. Ogni tessera `e associata ad una seconda tessera con cui andr`a a formare una parola bisillaba. Ad ogni parola `e associata una rappresentazione, quindi per esempio alla parola “cane” sar`a associata la figura di un cane, alla parola “mora” una mora e cos`ı via. Ogni immagine

`e divisa a met`a in due tessere rappresentanti le due sillabe componenti la parola. Il gioco consiste nel selezionare la coppia di tessere nella successione corretta per formare una parola di senso compiuto. Una volta accoppiate tutte le tessere della schermata il gioco sar`a completato. L’applicazione rac- coglie dati sulla velocit`a con cui il bambino compone una determinata parola e li invia via mail ad un indirizzo prescelto.

L’applicazione `e composta da poche semplici schermate. Nella prima schermata (figura 2.1) `e possibile scegliere di iniziare una partita in modalit`a singola o in modalit`a sfida, se si vuole giocare con un altro giocatore. Da questa schermata `e possibile accedere ad un’altra detta “di configurazione” (figura 2.2)che permette di modificare le impostazioni di gioco selezionando il numero di parole componibili in una schermata o turno, 2 o 3, il numero di turni di gioco, da 1 a 10, e l’indirizzo mail a cui inviare i risultati alla fine della partita. Le impostazioni predefinite per il numero di turni `e 8, per il numero di parole per turno `e 3 e per la mail `e il campo vuoto.

Selezionando la modalit`a gioco singolo, l’utente accede direttamente alla schermata di gioco (figura 2.3). Questa schermata `e divisa in due settori



Figura 2.1: Schermata iniziale

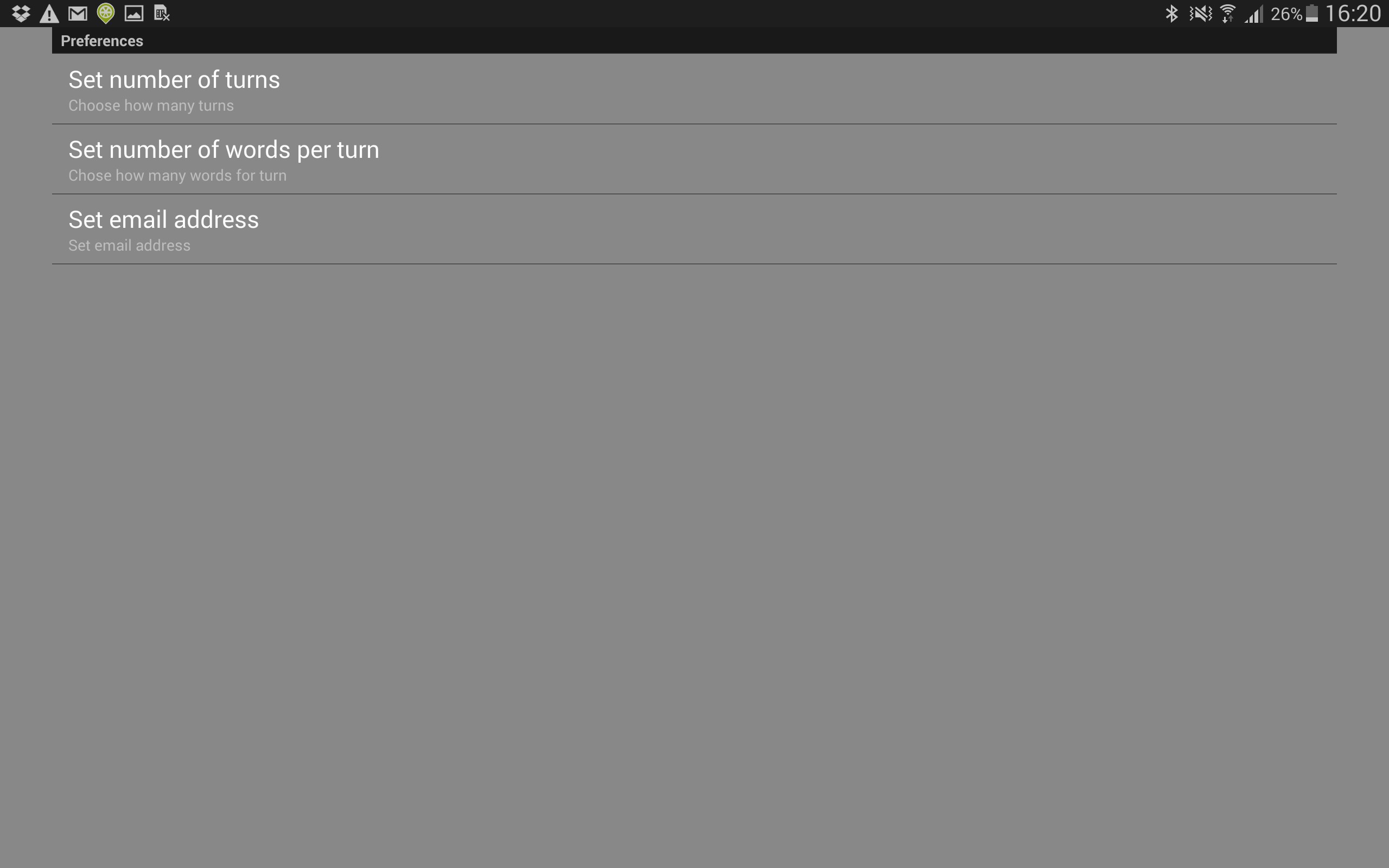


Figura 2.2: Schermata impostazioni

principali ovvero una sezione a sinistra, che presenta 3 tessere vuote, e una sezione a destra, con le tessere delle parole da comporre. Per comporre le parole l’utente selezioner`a le sillabe cliccando sopra ciascuna tessera. Quando si seleziona una tessera si pu`o ascoltare il suono della sillaba associata. Le parole devono essere composte nell’ordine corretto, prima-sillaba seconda-si llaba. Se l’utente seleziona per prima una prima sillaba questa comincia a lampeggiare, indicando univocamente quale parola deve essere completata. Se l’utente seleziona correttamente la seconda sillaba la parola `e completata e si sposta nella parte sinistra dello schermo, altrimenti la prima sillaba continua a lampeggiare. Ogni volta che una tessera viene selezionata viene pronunciata la sillaba corrispondente.

Al momento in cui la parola viene completata l’applicazione pronuncia la parola completa. L’utente la pu`o riascoltare selezionando la tessera nel campo risultati, oppure pu`o selezionare la bandiera inglese accanto alla stessa tessera che permette di ascoltare la stessa parola in inglese.

Quando il giocatore completa tutte le parole, ovvero quando la parte sinistra dello schermo `e completa, sulla parte destra comparir`a un bottone che permette di passare al turno successivo o, se le parole sono state tutte completate, permette di terminare la partita. Una volta terminata la partita l’applicazione riporta l’utente alla schermata iniziale.

Se viene selezionata la modalit`a di gioco “sfida” si accede invece ad una schermata di impostazione della connessione (figura 2.4). La connessione con altri giocatori avviene tramite bluetooth.

Accedendo alla schermata di impostazioni della partita doppia (figura 2.5) verr`a richiesto all’utente il permesso di rendersi visibile agli altri dispositivi per 5 minuti. I dispositivi con cui possiamo interagire si dividono in due categorie: la prima `e formata dai dispositivi con cui ci siamo connessi gi`a in

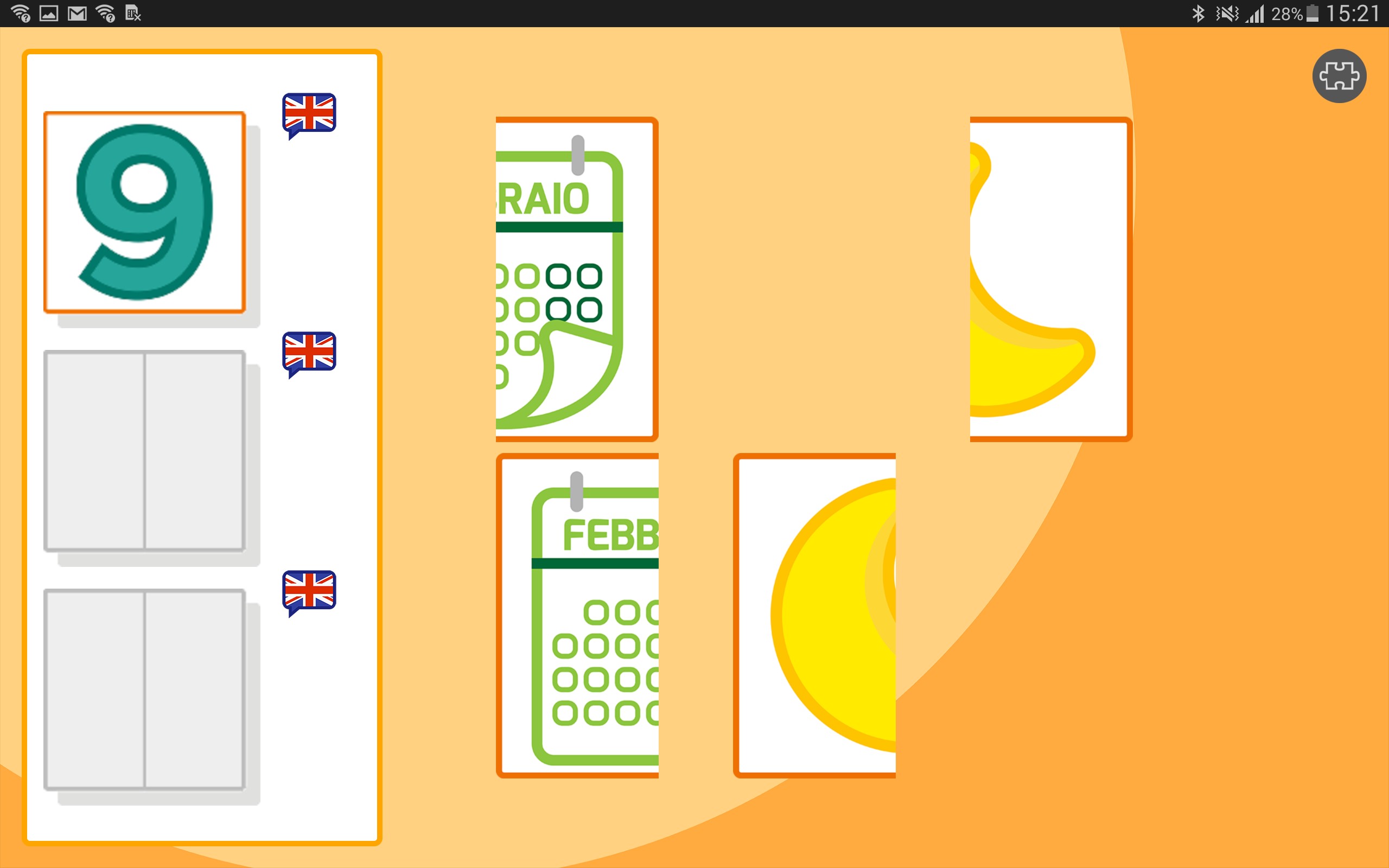


Figura 2.3: Schermata di gioco

passato, la seconda da quei dispositivi con cui non abbiamo mai interagito. Dando il permesso alla richiesta del sistema, ci si rende visibili a quei disposi- tivi con cui non abbiamo mai parlato; `e quindi una operazione fondamenta le quando dobbiamo instaurare una connessione con un giocatore per la prima volta.

Perch`e due giocatori possano giocare insieme occorre che entrambi sele- zionino la modalit`a di gioco sfida. Una volta che entrambi hanno raggiunto la schermata di impostazioni della partita doppia uno dei due invita il secondo a giocare. Cliccando il bottone centrale si ha accesso alla lista di disposi- tivi riconosci uti. Se la lista `e vuota o l’altro giocatore non compare nella lista baster`a cliccare sul bottone “scan” per dare avvio alla ricerca di nuovi dispositivi. Una volta che l’altro giocatore compare nella lista di possibili sfidanti, chi imposta la partita lo invita a giocare. A questo punto la partita

doppia ha inizio, l’applicazione mostrer`a la schermata di gioco ad entrambi i giocatori e l’invitato potr`a eseguire la prima mossa. I turni sono alternati e indicati da un semaforo in basso a destra. La composizione delle paro- le `e collaborativa, quindi il primo giocatore selezioner`a la prima sillaba e il secondo dovr`a completare la parola selezionat a e successivamente scegliere la prima sillaba della seconda parola. Il gioco continua fino all’esaurimento delle parole sullo schermo.

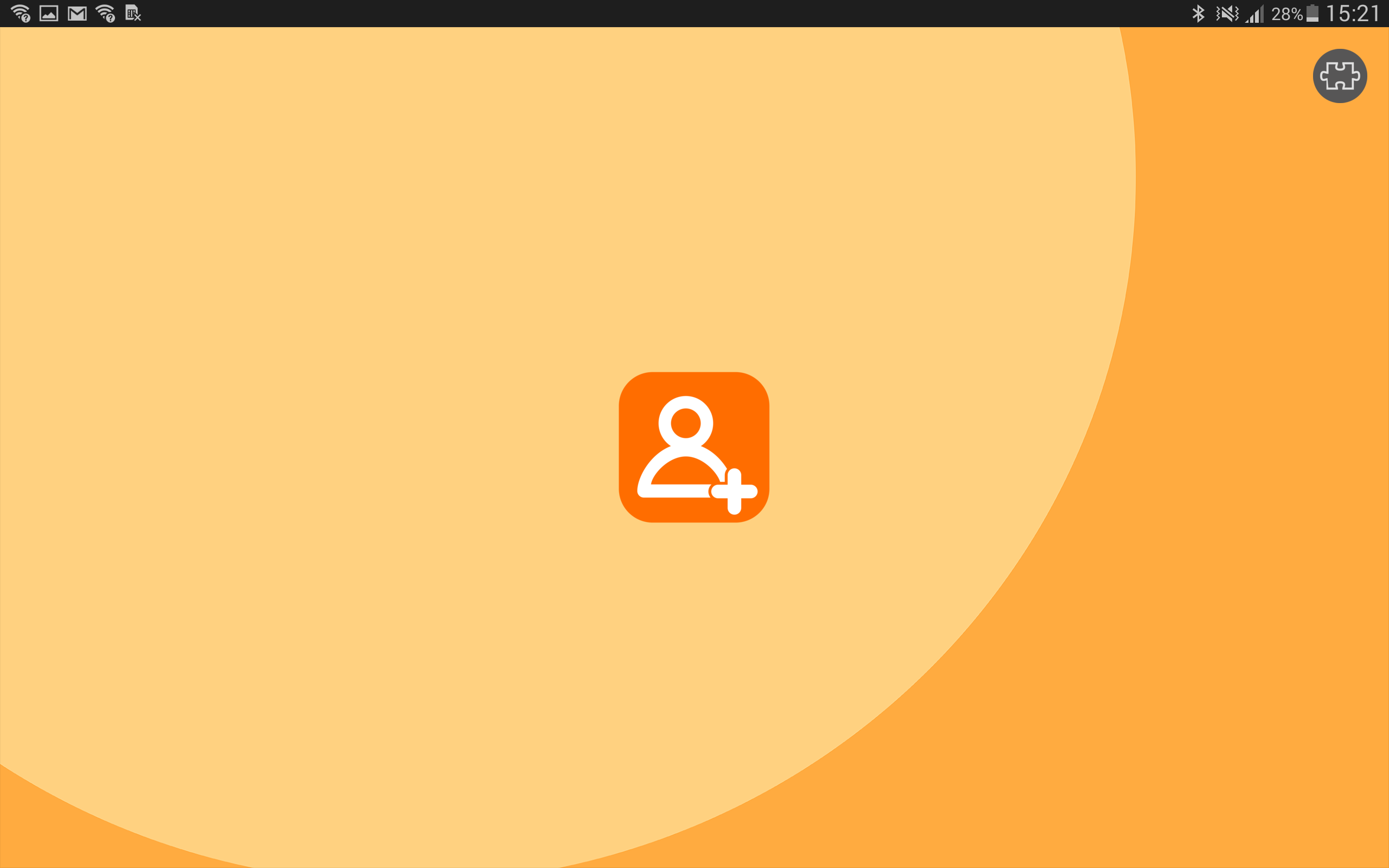


Figura 2.4: Schermata connection setup

L’applicazione LegoDigitalSonoro1 permette di raccogliere i dati sulla partita appena giocata. Ogni volta che viene completata una partita in mo- dalit`a giocatore singolo, viene inviata una mail all’indirizzo specificato nelle impostazioni contenente il numero di parole totale, tutte le parole completate e, per ogni parola, il tempo di completamento.



Figura 2.5: Schermata impostazione della partita doppia: lista dei giocatori disponibili

**2.2 Manuale Sviluppatore**

L’applicazione LegoDigitalSonoro `e stata progettata per girare su Android

4.2. I dati sono incorporati all’interno dell’applicazione, quindi non occorre scaricare da remoto alcuna informazione. L’unico contatto con un sistema remoto avviene al momento della inizializzazione della partita in modalit`a “sfida”

L’applicazione `e costituita da 3 activity principali che sono la MainActivity, che controlla la schermata inziale e permette all’utente di scegliere fra par- tita singola, multipla o modifica delle impostazioni, la MatchActivity, che controlla la schermata di gioco, e la ConnectionSetupActivity, che gestisce

la schermata di setup della partita in modalit`a doppio giocatore. Per un diagramma di interazione tra le *activity* citate, si veda figura 2.6.

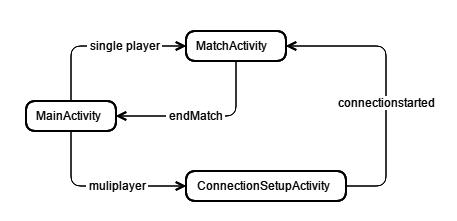


Figura 2.6: Diagramma delle Actvity

I dati sono costituiti da un file *txt* inserito nella cartella degli assets, che contiene su ogni riga l’informazione per una determinata parola. I dati sono composti da una stringa che contiene la parola in italiano, due stringhe, una per ciascuna sillaba, e una stringa che costituisce la traduzione in inglese della parola stessa. Il file viene letto una volta in fase di inizializzazione della partita. Altri dati sono costititi dalle immagini associate alle parole e dalle immagini relative a bottoni e agli altri elementi grafici, contenute nelle cartelle *drawable*.

Per gestire il dato “parola”abbiamo creato il tipo di dato Word di cui fanno parte una stringa per il nome, due istanze del tipo di dato Syllable e una stringa per la traduzione della parola in inglese (si veda figura 2.7). La risorsa grafica associata alla parola `e identificata dalla stringa della parola che rappresenta.

La partita `e regolata dalla MatchActivity che al momento della inizia- lizzazione recupera le informazioni sulla partita dalle *preferences* (numero di turni e numero di parole per turno), controlla la modalit`a di gioco (singo-

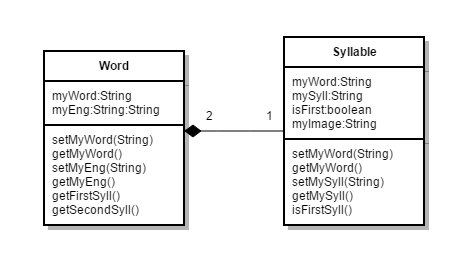


Figura 2.7: Class diagram: Word e Syllable

la o sfida) e instanzia il MatchManager. Il MatchManager gestisce la logica della partita, inizializza il gioco e ne modifica lo stato in base all’input del- l’utente. Il compito del MatchManager in fase di inizializzazione `e quello di instanziare il WordManager che a sua volta instanzia un WordReader per creare le parole e divide le parole nei diversi turni. La MatchActivity pos- siede inoltre due fragment che controllano lo stato dei due settori di cui `e composta l’interfaccia grafica. Un fragment `e chiamato ResultFragment e contiene le informazioni riguardanti le parole indovinate, l’altro `e invece il PlaygroundFragment che gestisce la griglia in cui sono inserite le tessere. Dato che il MatchManager contiene le informazioni sullo stato della partita avr`a un elenco per le parole presenti sulla schermata, uno per le sillabe nel fragment PlaygroundFragment e uno per i risultati nel ResultFragment. La MatchActivity implementa l’interfaccia ResultCallback, che permet- te di osservare le scelte dell’utente nel PlaygroundFragment e modificare di conseguenza il ResultFragment.

Per un *Class Diagram* delle classi appena citate, si veda figura 2.8.

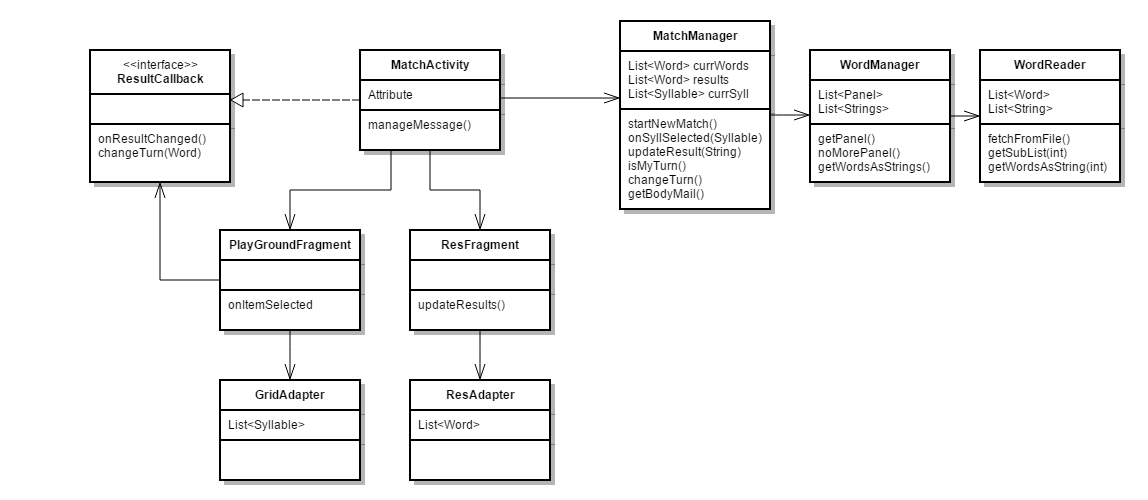


Figura 2.8: Class Diagram: Classi per la gestione dello stato della partita

**2.2.1 Partita singola**

Per inzializzare la partita singola il giocatore seleziona il bottone apposito nel-

la MainActivity. In questo modo viene chiamato il metodo startSingleMatch

che rimanda alla MatchActivity.

Nel caso di partita singola l’inizializzazione prevede che le parole vengano lette dal file di testo e raccolte dal WordReader. Il WordManager seleziona un sottoinsieme casuale delle parole del WordReader definito in base alle informazioni delle impostazioni (il numero di parole sar`a pari al prodotto fra numero di turni e numero di parole per turno). Le parole sono poi raccolte in tanti sottoinsiemi quanti sono i turni (si veda figura 2.9).

Una volta inizializzata la partita sullo schermo del giocatore appare una schermata con due sezioni, ognuna gestita da un apposito fragment. Le tessere sono inserite in una GridView, a cui `e associato un GridAdapter. Il PlaygroundFragment viene aggiornato ogni volta che l’utente seleziona una

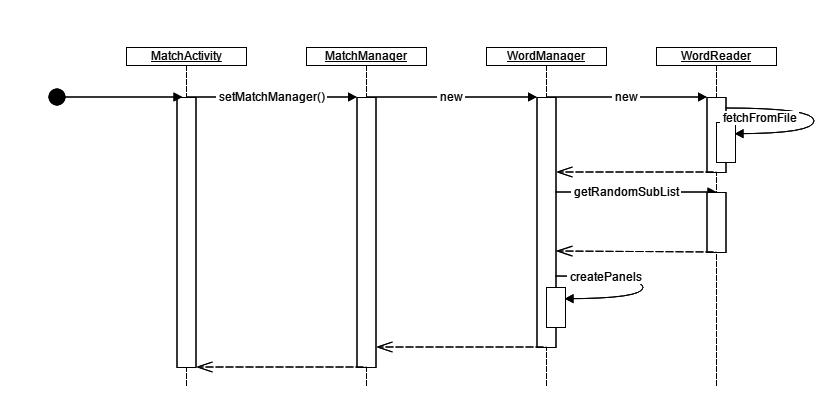


Figura 2.9: Sequence Diagram: inizializzazione della partita gioco singolo

delle tessere, e la tessera selezionata viene comunicata al MatchManager che in base a questa informazione aggiorna lo stato del gioco.

Il MatchManager in base alla tessera selezionata, ovvero alla sillaba, pu`o avere diversi comportamenti. Se la sillaba selezionata non completa nessuna parola ma `e la prima prima sillaba valida ad essere selezionata, viene aggior- nata l’ultima parola valida, se la sillaba selezionata coincide con la seconda sillaba dell’ultima parola valida viene aggiornato l’elenco dei risultati e viene notificato il ResultFragment, se la sillaba non `e valida, ovvero `e la prima ad essere selezionata ma non `e la prima di una parola, o ancora non com- pleta l’ultima parola valida, il MatchManager non aggiorna nulla. Abbiamo scelto, in questo caso, di non aggiornare l’ultima parola valida se il giocatore seleziona una nuova prima sillaba dopo la prima sillaba valida registrata. In questo modo al giocatore sar`a chiaro che per passare al completamento di una nuova parola `e necessario completare la parola precedentemente iniziata (*Sequence Diagram* in figura 2.10).

Quando l’utente seleziona una tessera che completa la parola valida, il

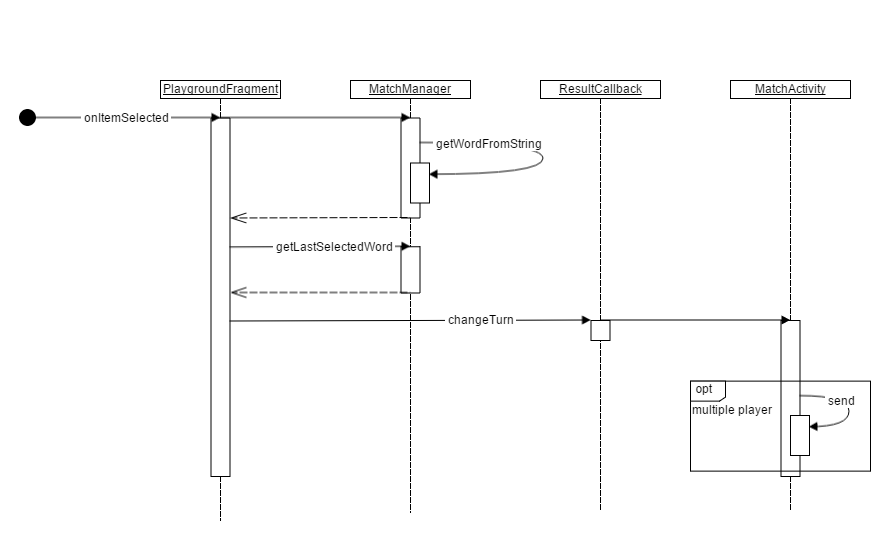


Figura 2.10: Sequence Diagram: selezione della prima sillaba di una parola, in questo caso non verr`a eseguita la parte nel riquadro opzionale dal momento che stiamo considerando la modalit`a di gioco singolo giocatore

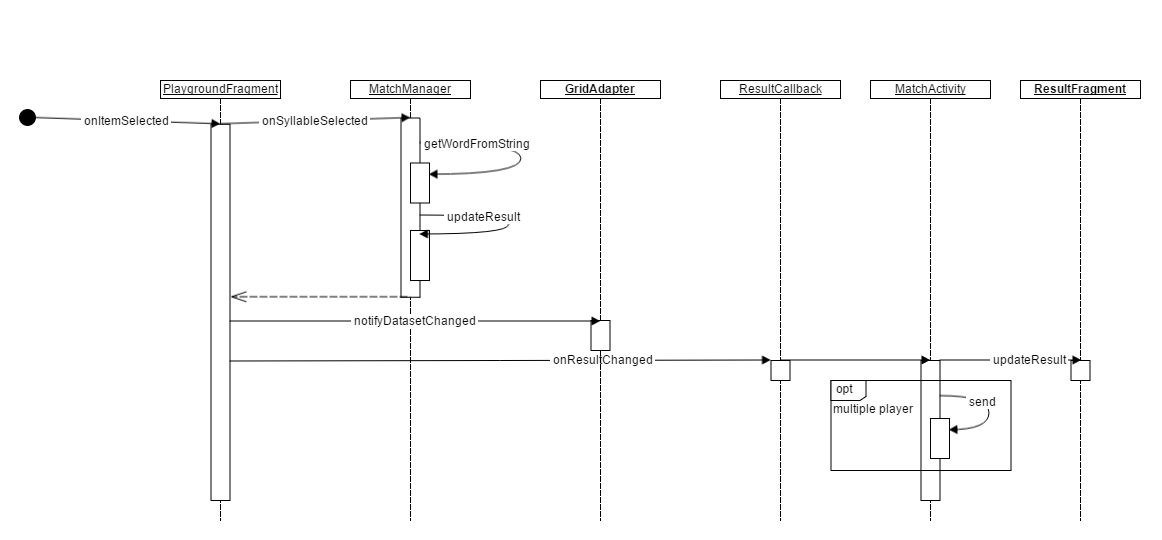


Figura 2.11: Sequence Diagram: completamento di una parola

PlaygroundFragment lo notifica al ResultFragment che aggiorna il conte- nuto della ListView. Se le parole sono tutte complete la MatchActivity rende visibile e cliccabile il tasto next per passare al turno successivo, ovve-

ro recuperare le parole del turno successivo e aggiornare il contenuto della

GridView. Se le non ci sono piu`

nuovi turni, ovvero il WordManager resti-

tuisce un turno vuoto, la partita si considera conclusa, l’utente visualizza un messaggio conclusivo e torna alla schermata iniziale (*Sequence Diagram in figura 2.11*).

**2.2.2 Partita multipla**

Per l’inizializzazione della partita multipla occorre mettere in comunicazione i dispositivi dei due giocatori tramite connessione Bluetooth. Questa operazio-

ne viene messa in atto facendo partire la connessione dalla ConnectionSetupActivity. Piu` precisamente nel metodo onCreate della ConnectionSetupActivity l’o-

perazione che viene eseguita `e quella di rendersi visibili, o *discoverable*, per eventuali nuove connessioni con nuovi giocatori. La connessione tramite Blue- tooth avviene in modalit`a Master-Slave, cio`e entrambi i dispositivi si mettono in ascolto di nuove connessioni ma uno dei due, in particolare quello che se- leziona il tasto apposito nella schermata, si connette all’altro e inizializza la partita.

Il giocatore che instaura la connessione esegue gli stessi passi del caso

partita in modalit`a singola, ma in piu`

invia le parole all’altro giocatore in

formato testuale. Il secondo giocatore, in qualit`a di slave, inizializza la partita solo quando riceve un messaggio dal master contenente tutte le parole. Nella pratica il gioco lato slave viene inizializzato come se le parole venissero lette dal file *txt*, ma vengono lette dal messaggio ricevuto dal master invece che da un file di configurazione. Il master inoltre invia allo slave anche il numero di

turni e il numero di parole per turno. Le parole vengono inviate esattamente come vengono lette dal file di testo, con la differenza che il separatore per le linee nel messaggio viene tradotto nel carattere “-” (*Sequence Diagram* in figura 2.12).

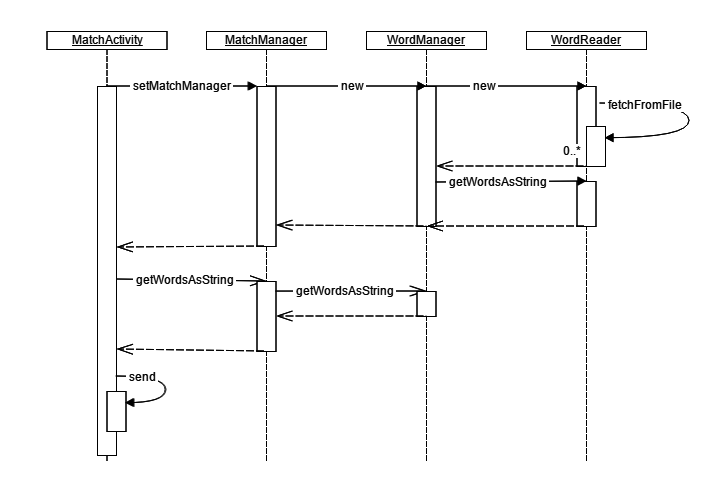


Figura 2.12: Sequence Diagram: inizializzazione giocatore master

Il gioco `e alternato quindi ogni giocatore sceglie durante il proprio turno una sillaba. A cominciare `e sempre il giocatore invitato, cio`e lo slave. Quando il primo giocatore sceglie una sillaba valida, all’altro giocatore verr`a inviato un messaggio con la stringa della parola selezionata. Il secondo giocatore sceglie la seconda sillaba e, se questa `e corretta, la stringa viene rimandata al primo giocatore. Questo verifica che la stringa ricevuta corrisponda all’ultima inviata e aggiorna sia il contenuto della GridView che quello della ListView.

La comunicazione fra master e slave `e codificata in tre diversi tipi di

messaggio, ad eccezione del primo contenente le parole dell’intera partita che

viene inviato unicamente dal master allo slave:

1. messaggio next: indica che le parole del turno sono state tutte comple- tate e il giocatore ha selezionato il tasto che porta al turno successivo;

2. messaggio end: indica che le parole della partita sono state completate e uno dei due giocatori ha selezionato il tasto che porta alla schermata successiva;

3. messaggio parola-valida: contiene la parola che il giocatore ha ini- ziato o completato e viene trattato, come visto, a seconda dello stato del gioco. Quando il gioco viene concluso appare sulla schermata un avviso che rimanda i giocatori alla schermata iniziale.

**Capitolo 3**

**Lego Digital Sonoro 2**

*Documentazione a cura di Giacomo Bresciani* (giacomo.bresciani@polimi.it)

**Indice**

3.1 Manuale Sviluppatore . . . . . . . . . . . . . . . . . . . . . . . 36

3.1.1 Introduzione . . . . . . . . . . . . . . . . . . . . . . . . 36

3.1.2 Database . . . . . . . . . . . . . . . . . . . . . . . . . 37

3.1.3 Logica di gioco . . . . . . . . . . . . . . . . . . . . . . 40

3.1.4 User Interface . . . . . . . . . . . . . . . . . . . . . . . 45

3.1.5 Librerie esterne . . . . . . . . . . . . . . . . . . . . . . 47

3.2 Manuale Utente . . . . . . . . . . . . . . . . . . . . . . . . . . 48

**3.1 Manuale Sviluppatore**

**3.1.1 Introduzione**

**Scopo del documento**

Lo scopo di questo documento `e quello di mostrare le scelte implementative fatte durante lo sviluppo dell’applicazione *Lego Digital Sonoro 2*. Esso si divide in 3 sezioni: *Database*, *Logica di gioco*, e *Interfaccia Utente*.

**Descrizione dell’applicazione**

*Lego Digital Sonoro 2* `e un videogioco per piattaforma Android progettato esclusivamente per dispositivi tablet. Lo scopo del gioco `e quello di compor- re parole a partire da tessere colorate rappresentanti diverse sillabe. Ogni tessera ha un colore associato direttamente alla sillaba che rappresenta e non appena viene toccata il dispositivo riproduce (tramite il sintetizzatore voca- le) il suono della sillaba. Toccando due tessere una dopo l’altra `e possibile comporre una parola che, se corretta, verr`a aggiunta all’elenco delle parole trovate sotto forma di immagine composta dalle due tessere. La schermata consente inoltre di riascoltare le parole trovate, sia in italiano sia in inglese. Una partita pu`o essere composta da piu` schermata; ogni schermata termina quando sono state trovate tutte le parole componibili con le sillabe mostrate (2 o 4).

Durante il corso della partita l’applicazione registra i tempi in cui vengono identificate le parole, e le invia ad una mail configurabile nelle impostazioni. E’ presente inoltre una modalit`a cooperativa a due giocatori a turni. Ogni turno consente ad un giocatore un tentativo per indovinare una parola.

La figura 3.1.1 mostra l’interfaccia di gioco.

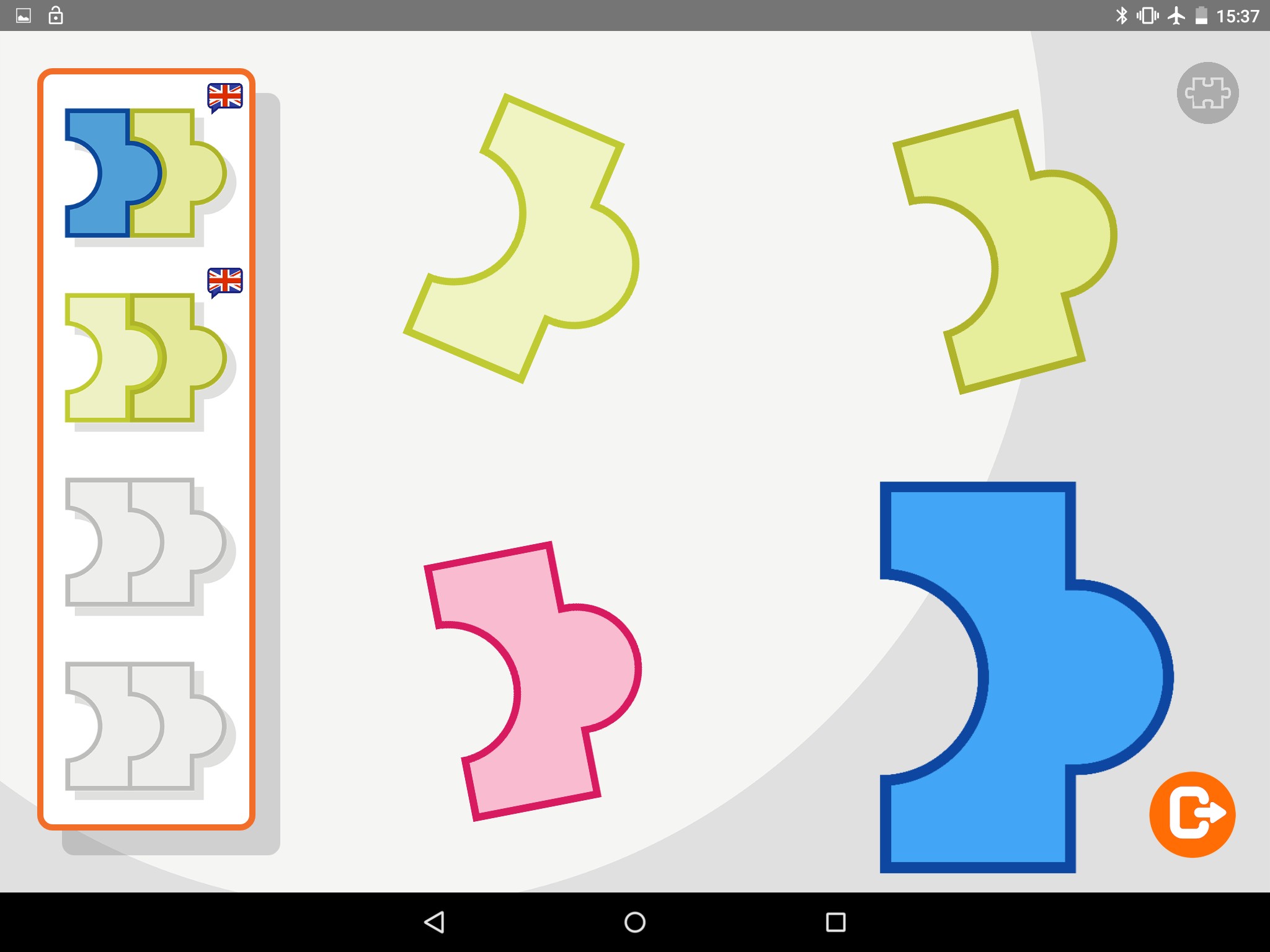


Figura 3.1: Schermata di gioco

**3.1.2 Database**

**Struttura**

Il database SQLite `e stato implementato tramite SugarORM, un ORM (Object- Relational Mapping) sviluppato appositamente per Android. In questo modo

le entit`a del database sono direttamente mappate su classi Java, utilizzando il pattern DAO (Data Access Object).

Il database contiene 5 entit`a all’interno del package

it.gbresciani.legodigitalsonoro.model:

Word Contiene la parola, sia in italiano sia in inglese, e le sillabe che lo compongono.

Syllable Contiene la sillabla e l’esadecimale del colore associato.

WordStat Permette di tracciare il momento in cui una parola viene trovata.

GameStat Contiene le statistiche di una particolare partita

In figura 3.1.2 sono riportati dettagli delle entit`a.

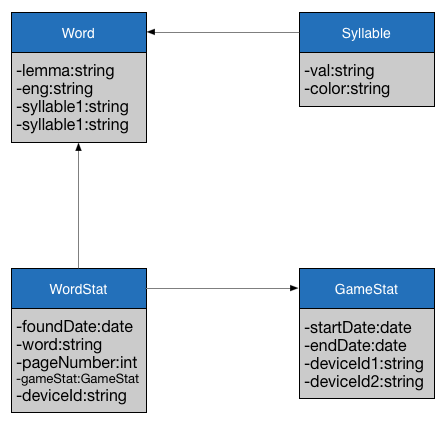


Figura 3.2: Entit`a presenti nel database

Listing 3.1: Porzione del file words.json

[

{

"lemma":"arco", "eng":"arc", "syllable1":"ar", "syllable2":"co"

},

...

**Inizializzazione**

Il database viene inizializato la prima volta che l’applicazione viene aperta con i dati in formato *JSON* presenti nei files words.json e syllable.json contenuti nella cartella degli assets.

Il listing 3.1 contiene un estratto del file words.json

Il mapping degli elementi JSON con la classe Java associata avviene tra- mite la libreria Open Source GSON, sviluppata da Goolge, che fa uso di annotazioni per mappare gli attributi delle classi con i campi JSON. L’operazione di conversione dei dati da JSON a classi Java e il successivo inse- rimento all’interno del database, in quanto operazioni relativamente lunghe, non vengono eseguite all’interno del *Thread* principale ma in uno differente grazie ad un IntentService contenuto in it.gbresciani.legodigitalsonoro.services.InitDBService.

**3.1.3 Logica di gioco**

La logica di gioco `e affidata all’Activity PlayActivity; essa gestisce inoltre i Fragments (vedi capitolo 3.1.4) che visualizzano l’Intefaccia Utente.

**Glossario**

In questa sezione vengono descritti acuni termini utili alla comprensione delle successive sezioni.

**Page** Una partita `e composta da una o piu` pagine. Una pagina pu`o conte- nere 2 o 4 sillable e da 1 a 4 parole da trovare

**User role** Durante una partita multi player i giocatori si distinguono in “master”, colui che “orchestra” la partita, e “slave”, colui che vi par- tecipa in modo passivo

**Turno** Durante una partita multi player rappresenta la possibilit`a per un giocatore di provare a indovinare una parola. Il giocatore che non `e in possesso del turno attende la mossa dell’altro

**Event Bus**

Per far comunicare i vari componenti dell’applicazione (Activities, Fragments,

Services) `e stato utilizzato Otto, un *Event Bus*, sviluppaato appositamente per Android, che permette ai vari componenti di postare eventi su un unico Bus e per riceverli in modo asincrono.

La classe BusProvider permette, tramite un metodo statico, di accedere allo stesso Bus in ogni punto dell’applicazione.

All’interno del package packageit.gbresciani.legodigitalsonoro.events

son presenti numerose classi (ad esempio WordClickedEvent o PageCompletedEvent)

che rappresentano i diversi eventi generati e “ascoltati” dai componenti del- l’app.

La classe NineBus `e una piccola estensione del Bus predefinito di Otto che permette di pubblicare tutti gli eventi sul MainThread, anche se generati in un altro Thread.

**GameState**

GameState `e la classe che rappresenta lo stato di una partita, sia essa single o multi player. Viene inizializzato da PlayActivity, e aggiornato durante il corso della partita in seguito ai diversi eventi prodotti dagli altri componenti dell’app.

I suoi attributi sono:

**pageNumber** Il numero di pagina corrente.

**pages** Il numero di pagine totali della partita in corso (stabilito nelle impo- stazioni).

**pageWordsToFindNum** Il numero di parole ancora da trovare nella pagi- na corrente.

**wordsAvailable** Un ArrayList contentente le parole ancora da trovare.

**wordsFound** Un ArrayList contenente le parole gia trovate.

**syllables** Un ArrayList contenente le syllabe presenti nella pagina corrente.

**currentPlayer** Se la partita `e multi player contiene una stringa che rappre- senta il ruolo del giocatore che detiene in quel momento il turno.

**currentPlayerDeviceId** Se la partita `e multi player contiene una stringa che rappresenta l’id del dispositivo che detiene in quel momento il turno.

La classe GameState contiene attributi annotati con @SerializedName in modo da poter essere convertita in un oggetto JSON (tramite GSON) ed essere inviata al secondo giocatore nel caso di partita multiplayer (maggiori dettagli nella sezione 3.1.3).

**Statistiche di gioco**

Le satistiche della partita, come accennato nel capitolo sul database, vengono salvate tramite i DAO WordStat e GameStat che contengono, ad esempio, i tempi di ritrovamento delle parole, le parole stesse, la pagine su cui sono state trovate o eventualmente l’id del giocatore che le ha trovate.

**Game flow**

Tutta le fasi di una partita sono gestite dall’Activity PlayActivity. All’av- vio, cio`e durante la chiamata al metodo onCreate() essa esegue le seguenti operazioni:

1. Carica i settaggi della partita

2. Carica i suoni di gioco

3. Istanzia il sintetizzatore vocale TextToSpeech

4. In base al tipo di partita (single o multi player) inizializza o meno il

Bluetooth e setta la variabile multi a “MASTER” o “SLAVE” I principali metodi di PlayActivity sono:

startGame() Si occupa di inizializzare gli oggetti per le statistiche e chiama

constructPage() and startPage()

constructPage() Inizializza lo stato della partita (GameState) con i da- ti relativi alla pagina corrente. Chiama i metodi accessori Helper. chooseSyllables() (sceglie in modo casuale le sillabe per la pagina) e Helper.permuteSyllablesInWords() (calcola le permutazione delle sillabe scelte per trovare le parole componibili)

startPage() Inizializza i Fragments che mostreranno l’interfaccia utente

updateState() Si occupa di gestire la logica di gioco: determina se tutte le parole sono state trovate o se interrompere il gioco quando non `e il proprio turno e posta sul Bus l’evento StateUpdatedEvent in modo che i Fragments possano reagire correttamente

L’Activity presenta inoltre diversi metodi annotati con @Subscribe che permettono di reagire agli eventi pubblicati da altri componenti.

**Invio statistiche**

L’invio delle statistiche avviene tramite l’invio di una mail all’indirizzo set- tato nelle impostazioni dell’applicazione.

L’invio `e gestito dall’IntentService GenericIntentService al quale viene passato l’id corrispondente alla partita appena conclusa. Interrogando il da- tabase esso carica i dati e li inserisce nel corpo dell’email formattandoli come mostrato nel listing 3.2. Il listing 3.3 ne mostra invece un esempio.

**Multiplayer**

Come gi`a detto la modalit`a multi player segue le stesse regole di quella single player se non per il fatto che i due giocatori provano a turno a trovare una parola.

Il tablet “master” `e quello che inizia il processo di connessione scegliendo,

Listing 3.2: Struttura delle satistiche inviate

Dispositivo N: device id

Tempo totale: mm:ss

N - word: mm:ss (# of page on which the word was found)

...

- One player -

Listing 3.3: Esempio di statistiche

Dispositivo 1: BC:20:A4:73:6B:49

Tempo totale: 00:13

1 - fumo: 00:02 (1)

1 - muro: 00:13 (2)

- Two Player -

Dispositivo 1: BC:20:A4:73:6B:49

Dispositivo 2: D4:0B:1A:15:FD:AD Tempo totale: 00:14

2 - ceno: 00:02 (1)

1 - noce: 00:09 (1)

2 - buco: 00:14 (2)

tra i dispositivi in ascolto, il tablet “slave”. I vari aspetti della partita come i settaggi, le sillabe presenti e le parole da trovare vengono tutti stabiliti dal “master” che si occupa quindi anche di gestire il flow del gioco, mentre lo “slave” partecipa in modo passivo.

La comunicazione tra i due dispositivi avviene tramite Bluetooth. La classe che si occupa di stabilire la connessione `e BluetoothService; essa si serve di tre Thread differenti (AcceptThread, ConnectThread, ConnectedThread) per ricevere un connessione, effettuare una connessione, e inviare e ricevere messaggi.

I messaggi scambiati sono stringhe contenenti oggetti JSON preceduti da un header che ne identifica la natura. I possibili messaggi sono 4:

**GAME STATE** Contiene la rappresentazione JSON dell’oggetto GameState descritto nella sezione 3.1.3. Solo il “master” invia questo tipo di mes- saggio allo “slave”, il quale ne ricava tutte le informazioni necessarie.

**SIMPLE TURN PASS** Lo “slave” invia questo messaggio al “master” quando non `e riuscito a trovare parole e quindi sta semplicemente passando il turno.

**WORD FOUND** Lo “slave” invia questo messaggio insieme alla parola trovata, in modo che il “master” possa aggiornare lo stato della partita.

**GAME END** Il “master” invia questo messaggio allo “slave” al termine della partita.

**3.1.4 User Interface**

Come anticipato nel capitolo 3.1.3, l’interfaccia grafica del gioco `e gestita quasi interamente da Fragments, che rispondono agli eventi pubblicati sul

Bus dall’activity PlayActivity aggiornando i loro componenti.

I due principali Fragments sono WordsFragment e SyllablesFragment (vedi figura 3.1.4), che si occupano rispettivamente di gestire la porzione di schermo contenente le parole, e la porzione contenente le sillabe disponibili.

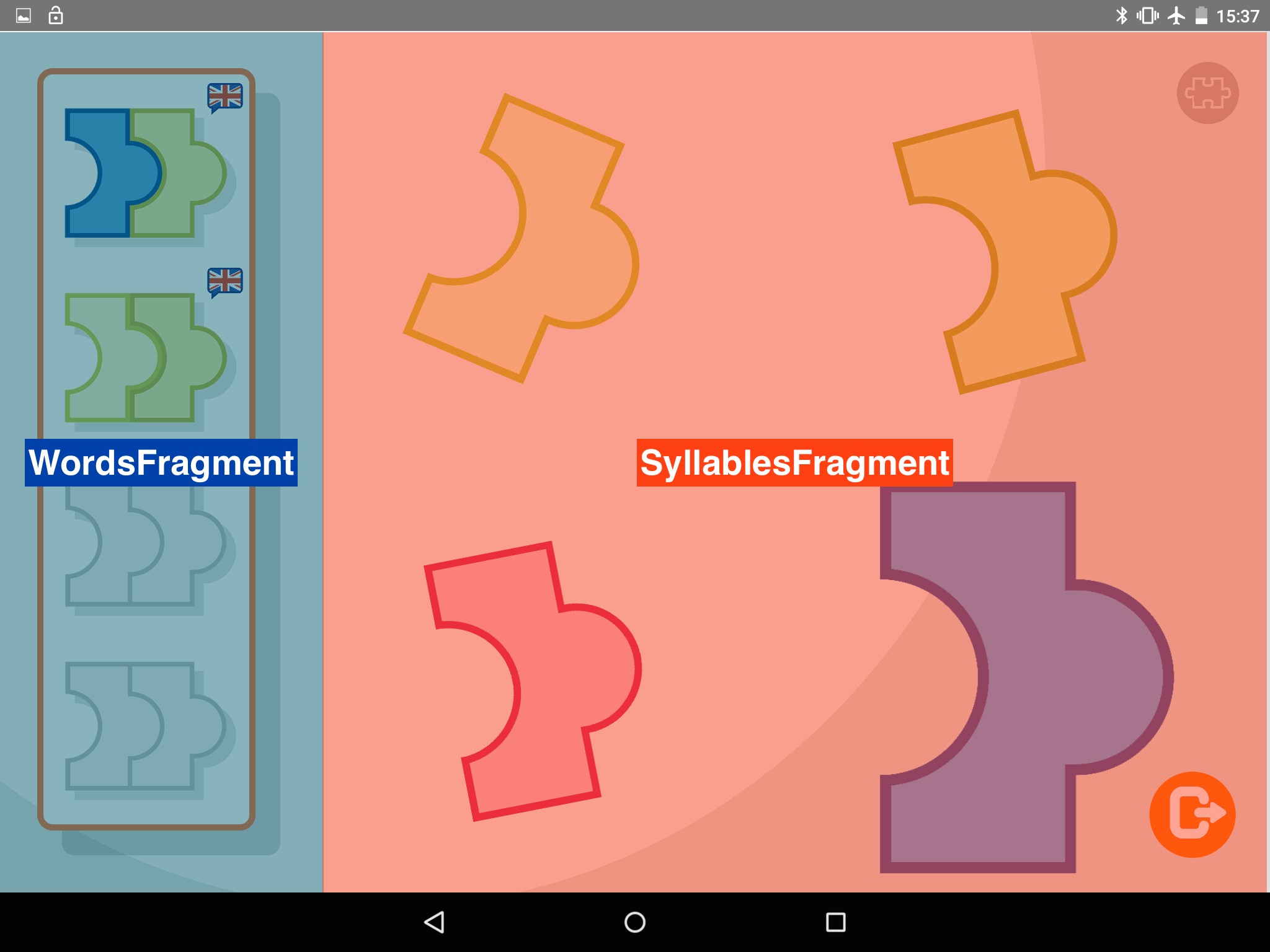


Figura 3.3: Fragments della schermata di gioco

**WordsFragment**

WordsFragment si occupa di gestire la parte di interfaccia grafica che rigurda le parole gi`a trovate e quella ancora da trovare.

Esso presenta una lista verticale di “slot” che possono essere “vuoti”, o “riem- piti” con le tessere di puzzle corrispondenti alle sillabe che formano la parola. Viene inoltre mostrata accanto ad ogni slot “pieno” una bandiera inglese che

se toccata permette di ascolare il suono della parola tradotta in inglese.

Il metodo initUI() si occupa, dato il numero di parole da trovare e la dimen- sione dello schermo (calcolata a runtime), di calcolare la dimensione degli slot corretta per gli slot con lo scompo di riempire al meglio lo spazio disponibile. Le immagini degli slot sono fornite come file PNG all’interno della cartella assets e vengono caricate dinamicamente come Bitmap della dimensione corretta (per risparimare memoria) dal metodo loadWordBitmap().

Il Fragment `e sottoscritto all’evento StateUpdatedEvent in modo da reagi-

re ai cambiamenti di stato della partita, come il ritrovamento di una nuova parola.

**SyllablesFragment**

SyllablesFragment si occupa di gestire la parte di interfaccia grafica che rigurda le sillabe disponibili nella pagina corrente.

Esso presenta una griglia sparsa di 2 o 4 tessere di puzzle colorate rappre- sentanti ognuna una sillaba. Se toccate le tessere vengono selezionate e si “alzano” per restituire un feedback della selezione e riproducono il suono della sillaba a cui sono associate.

Al pari di WordsFragment calcola la dimensione corretta delle immagini e le carica come Bitmap.

**3.1.5 Librerie esterne**

Di seguito vengono elencate le librerie utilizzate per lo sviluppo dell’applica- zione.

[**SugarORM**](http://satyan.github.io/sugar)

Libreria Open Source ORM.

[**Gson**](https://code.google.com/p/google-gson/)

Libreria Java che permette la conversione tra oggetti Java e la loro rappresentazione JSON.

[**Otto**](http://square.github.io/otto/)

Libreria che permette di sfruttare un event bus.

[**ButterKnife**](https://github.com/JakeWharton/butterknife)

Libreria per l’injection delle View.

**3.2 Manuale Utente**

Dalla schermata principale (figura 3.4), toccando l’icona grigia in alto a sini- stra dello schermo, si pu`o accedere alla schermata delle impostazioni (figura

3.5) da cui si possono impostare i parametri di gioco e l’indirizzo e-mail per l’invio dei dati.

Sempre dalla schermata iniziale si pu`o dare inizio al gioco sia in modalit`a multi-giocatore, che singolo. Se si sceglie la prima, allora si verr`a rimandati alla schermata di associazione tramite *bluetooth* (figura 3.6).

Una volta iniziata la partita (figura 3.7), si potr`a giocare cliccando sulle sillabe fino al suo termine.



Figura 3.4: La schermata principale

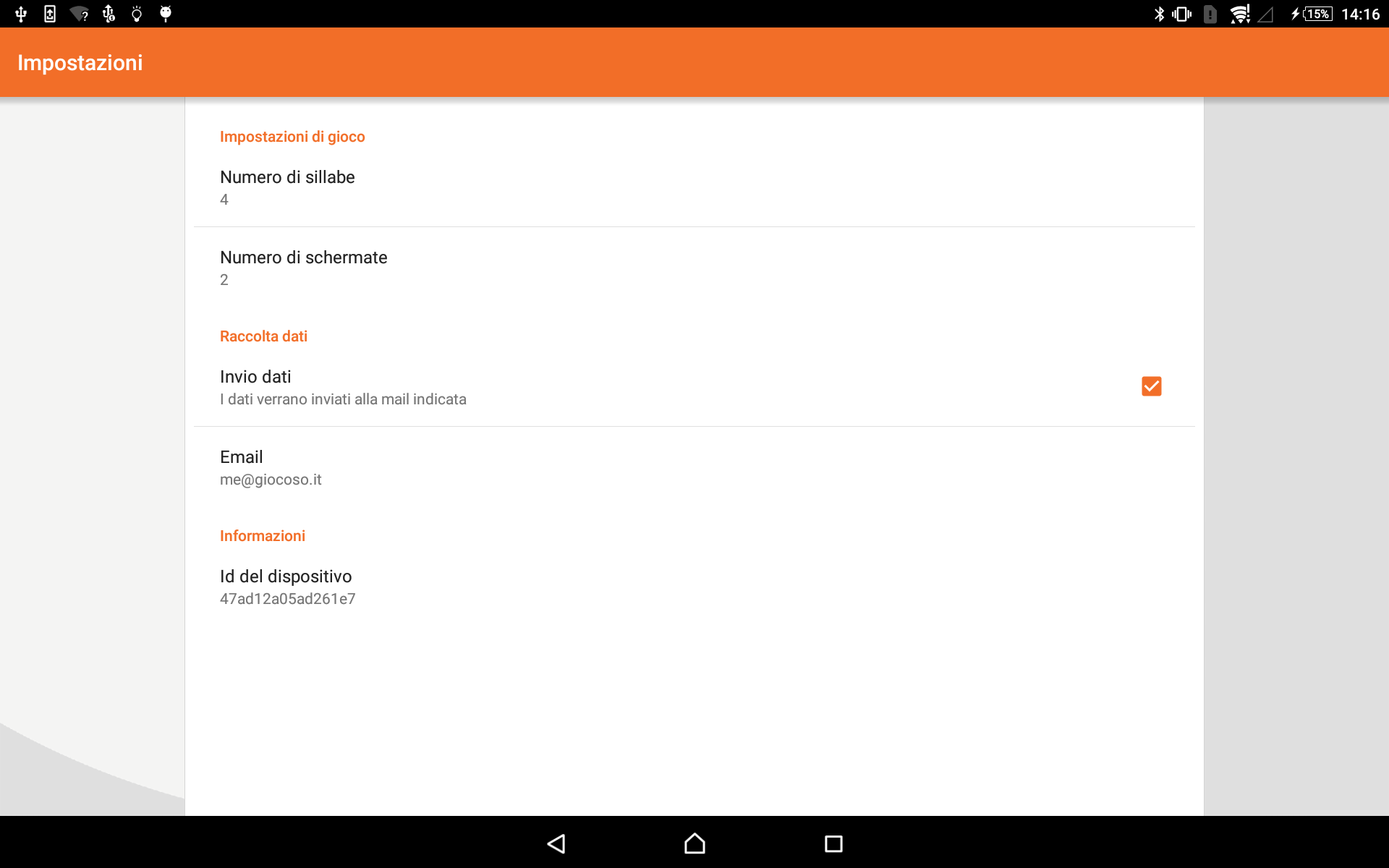


Figura 3.5: La schermata di impostazioni



Figura 3.6: La schermata di associazione tramite *bluetooth*

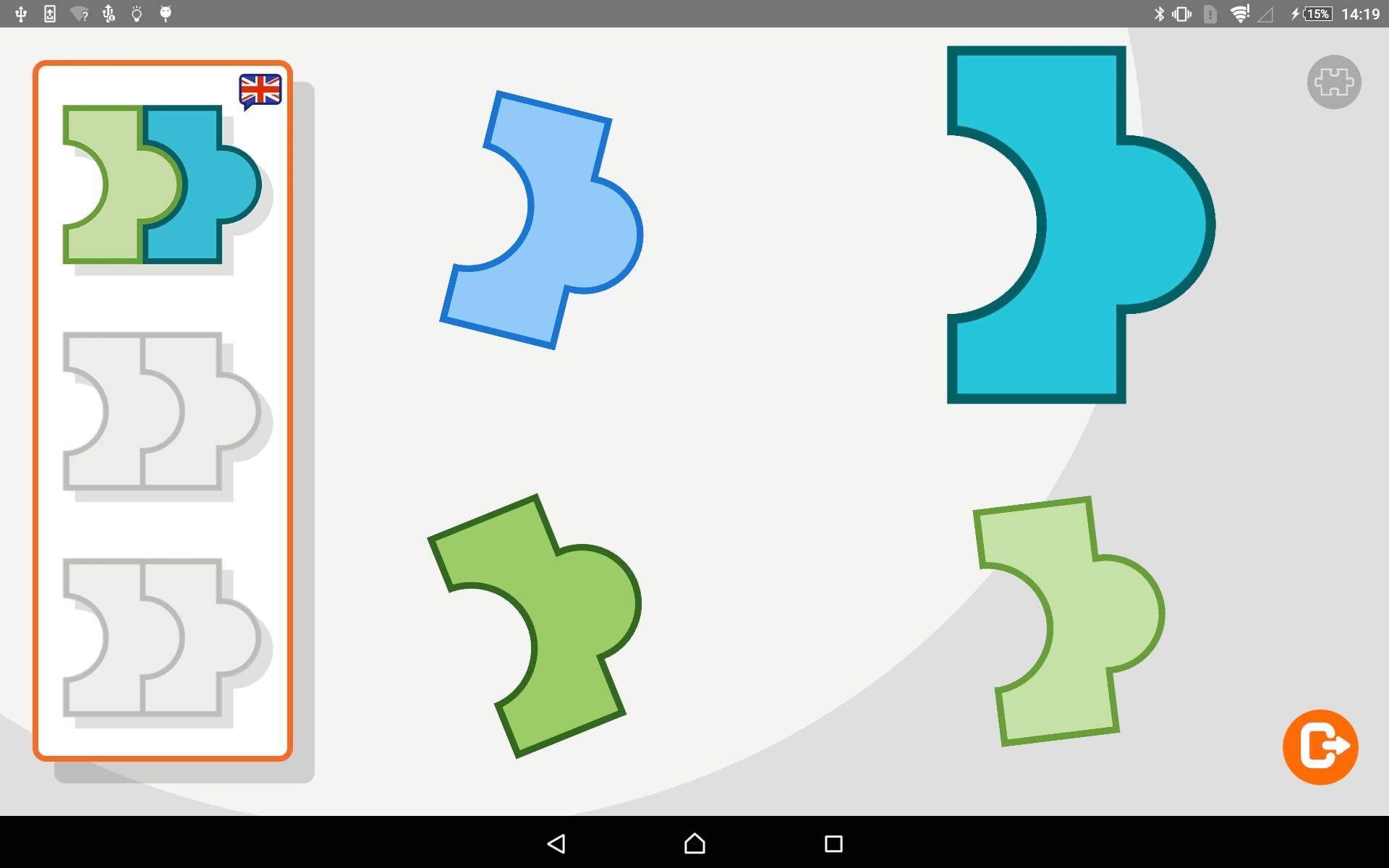


Figura 3.7: La schermata di gioco

**Capitolo 4**

**Trova l’Intruso 1**

*Documentazione a cura di Paolo Ferraris* (paolo2.ferraris@mail.polimi.it)

**Indice**

4.1 Manuale Sviluppatore . . . . . . . . . . . . . . . . . . . . . . . 53

4.1.1 Descrizione delle classi . . . . . . . . . . . . . . . . . . 53

4.1.2 Modello della sessione di gioco . . . . . . . . . . . . . . 59

4.2 Manuale Utente . . . . . . . . . . . . . . . . . . . . . . . . . . 61

**4.1 Manuale Sviluppatore**

**4.1.1 Descrizione delle classi**

Per prima cosa vengono descritte le classi principali, partendo dal modello che definisce il gioco, in seguito vengono descritte le classi Helper che permettono la gestione della logica, e infine le Activity utilizzate.

**Classi del modello dati**

Di seguito sono descritte le classi che gestiscono il modello del gioco, rappre- sentate nel *class diagram* della Figura 4.1:

**Game** `e la classe che gestisce la partita, i principali metodi sono:

*•* initialize(): metodo che inizializza la sessione di gioco, creando i livelli e assegnandone gli oggetti da visualizzare.

*•* initializeMultiplayerSession(booleanguest): metodo che sta- bilisce i turni di gioco nelle partite multiplayer, la variabile guest indica se il dispositivo `e quello che `e stato invitato a giocare.

*•* goToNextScreen(): metodo che attiva la schermata successiva e restituisce true. Nel caso ci si trovi nell’ultimo livello restituisce false.

*•* fillScreens(): riempie le schermate con le immagini in base alla categoria e al numero di oggetti da visualizzare.

*•* CreateGameDescription(Contextcontext): crea la descrizione della partita da mandare via mail.

*•* restart(): riavvia il gioco con le stesse impostazioni.

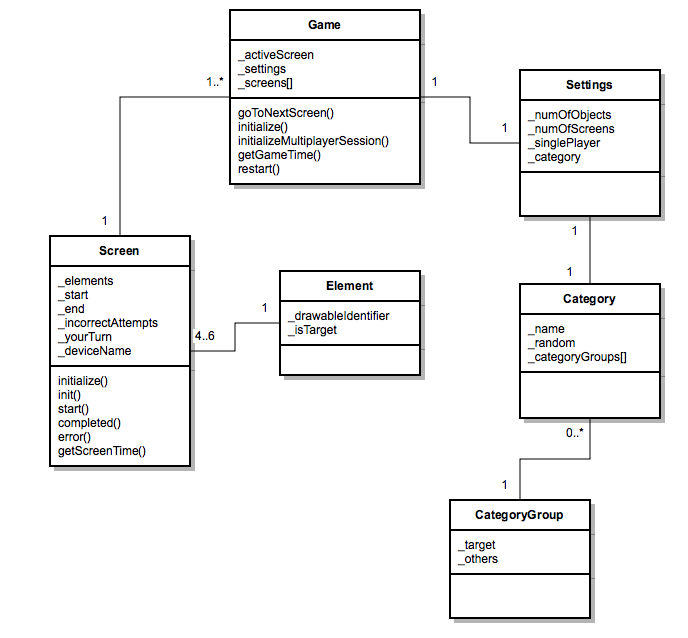


Figura 4.1: Class Diagram

**Settings** Classe che gestisce le impostazioni di gioco, possiede le seguenti propriet`a:

*•* \_singlePlayer(Boolean): indica la modalit`a di gioco.

*•* \_numOfObjects(int): indica il numero di immagini visualizzate per livello.

*•* \_numOfScreens(int): indica il numero di livelli impostati.

*•* \_category(Category): indica la categoria di immagini seleziona- ta.

**Screen** Classe che rappresenta il livello di gioco, possiede come propriet`a il tempo di inzio livello, il tempo di fine, una variabile booleana che indica se `e il turno del giocatore corrente, il numero di errori e un array contenente le immagini da visualizzare. I principali metodi sono:

*•* initialize(): inizializza il livello, inserendo le immagini.

*•* init(): azzera le variabili del livello, ossia numero di errori, tempo di gioco.

*•* start(): avvia il gioco su questo livello, settando la variabile

\_start\_time.

*•* error(): incrementa il conteggio degli errori di uno.

*•* completed(): metodo evocato quando il livello `e completato, setta la variabile \_end\_time.

*•* getScreenTime(): restituisce il tempo di gioco.

**Element** Classe che rappresenta un immagine, possiede due attributi, il no- me dell’immagine presente nella cartella drawable e un valore booleano che indica se `e l’elemento target.

**Category** Classe che rappresenta una categoria di immagini, possiede co- me propriet`a il nome della categoria, un campo booleano che indica se si tratta di una categoria “Casuale” e una lista di oggetti Category- Groups relativi alla categoria. Il costruttore accetta un JSONObject contenente la definizione della categoria e da esso inizializza l’oggetto.

**CategoryGroup** Classe che rappresenta una gruppo di immagini che deve essere rappresentato in una schermata. Definisce l’elemento target e i restanti elementi.

**Classi Helper**

**CategoryHelper** Gestisce il caricamento delle categorie di immagini dal file .json dove sono definite.

**GameHelper** Gestisce le meccaniche di gioco e la comunicazione tra i dispositivi in multiplayer. I principali metodi di questa classe sono:

*•* onMainActivityCreate(): metodo utilizzato all’avvio dell’acti- vity principale, avvia il riconoscimento dei dispositivi sul network (per le partite in multiplayer) e carica le impostazioni di gioco precedentemente salvate.

*•* onMainActivityDestroy(): chiamato quando l’activity princi- pale viene distrutta, viene terminato il servizio di discovery sul network e viene chiusa la connessione (se attiva).

*•* registerCurrentActivity(Activityactivity): metodo chia- mato nell’onCreate di ogni activity, registra l’activity mostrata a schermo per gestire al meglio le comunicazioni tra i dispositivi in multiplayer e la visualizzazione di avvisi o dei feedback di gioco.

*•* startGame(): Questo metodo inizializza il gioco, viene chiamato il metodo Game.initialize() e in seguito se il gioco `e in Single- Player viene caricata l’activity Screen nella quale viene mostrato il primo livello, altrimenti si passa all’activity MultiPlayerDisco- veryActivity per invitare un giocatore.

*•* quitGame(): questo metodo serve per terminare una partita, e nel caso di partite multiplayer viene chiusa anche la connessione.

*•* nextScreen(): metodo che serve per passare al livello successivo, in caso di partite multiplayer viene inviato l’analogo messaggio all’altro dispositivo.

**MultiPlayerServiceHelper** Gestisce il riconoscimento dei dispositivi che eseguono il gioco sulla rete per permettere l’avvio di partite multiplayer.

**ConnectionHelper** Il multiplayer `e implementato usando i socket Java, questa classe gestisce la connessione, l’invio di messaggi sul network e inizializza gli oggetti ServerHelper e ClientHelper.

**ServerHelper** Gestisce il lato server della connessione, avviando un SocketServer

thread che si occupa dell’assegnamento di una connessione al socket client appena arriva una richiesta al server.

**ClientHelper** Questa classe istanzia due thread, un thread che si occupa della ricezione dei messaggi in arrivo al socket utilizzato per la connes- sione, e uno che si occupa di inviare i messaggi al dispositivo col quale si sta giocando.

**Gestione della comunicazione in partite multiplayer**

Per la gestione delle comunicazioni tra dispositivi nelle partite multiplayer `e stata creata una classe GameMessage, la quale definisce una propriet`a di tipo Serializable, che rappresenta il contenuto del messaggio, e una propriet`a chiamata Type un enum che indica il tipo di messaggio, che pu`o assumere i seguenti valori:

*•* ConnectionRequest: i messaggi di questo tipo non hanno contenuto e servono ad inoltrare a un dispositivo una richiesta di connessione per avviare una partita;

*•* ConnectionAccepted: questo messaggio viene inviato dal client al server quando il giocatore accetta di giocare;

*•* ConnectionClosed: questo messaggio viene inviato prima di chiudere la connessione;

*•* SendGame: con questo messaggio viene inviata la struttura del gioco al dispositivo invitato a giocare;

*•* SendGameAck: questo messaggio conferma la ricezione del messaggio precedente da parte del client;

*•* StartGame: questo messaggio viene inviato al dispositivo client per comunicargli di avviare il gioco, alla sua ricezione il client avvia il gioco e manda il messaggio di ack al server;

*•* StartGameAck: questo messaggio conferma la ricezione del messaggio precedente, alla sua ricezione il dispositivo server a sua volta avvia il gioco;

*•* ElementPressed: con questo messaggio viene inviato l’indice dell’ele- mento premuto all’altro dispositivo in modo da visualizzare il feedback;

*•* NextScreen: con questo messaggio si comunica al dispositivo avversa- rio che il giocatore ha premuto il pulsante per cambiare schermata.

**Activity**

Di seguito vengono descritte le principali Activity presenti nell’app:

**MainActivity**: `e l’activity che si presenta all’avvio del gioco, inizializza la propriet`a globale GameHelper e permette di avviare il gioco.

**ScreenActivity**: l’activity che gestisce il livello di gioco, all’avvio vengono caricate le immagini e mostrate a schermo, in seguito si attende l’input dell’utente e si visualizzano a schermo i feedback.

**ResultsActivity**: l’activity che viene mostrata alla fine del gioco, mostra i risultati dei livelli e permette di riavviare il gioco oppure tornare alla MainActivity.

**MultiPlayerDiscoveryActivity**: in questa activity vengono mostrati gli altri giocatori e si pu`o iniziare una partita multiplayer.

**SettingsActivity** e **ConfigDeviceActivity** permettono la configurazione delle impostazioni della partita e del dispositivo.

**4.1.2 Modello della sessione di gioco**

Nella Figura 4.2 `e riportato l’activity diagram che schematizza lo svolgimento del gioco.

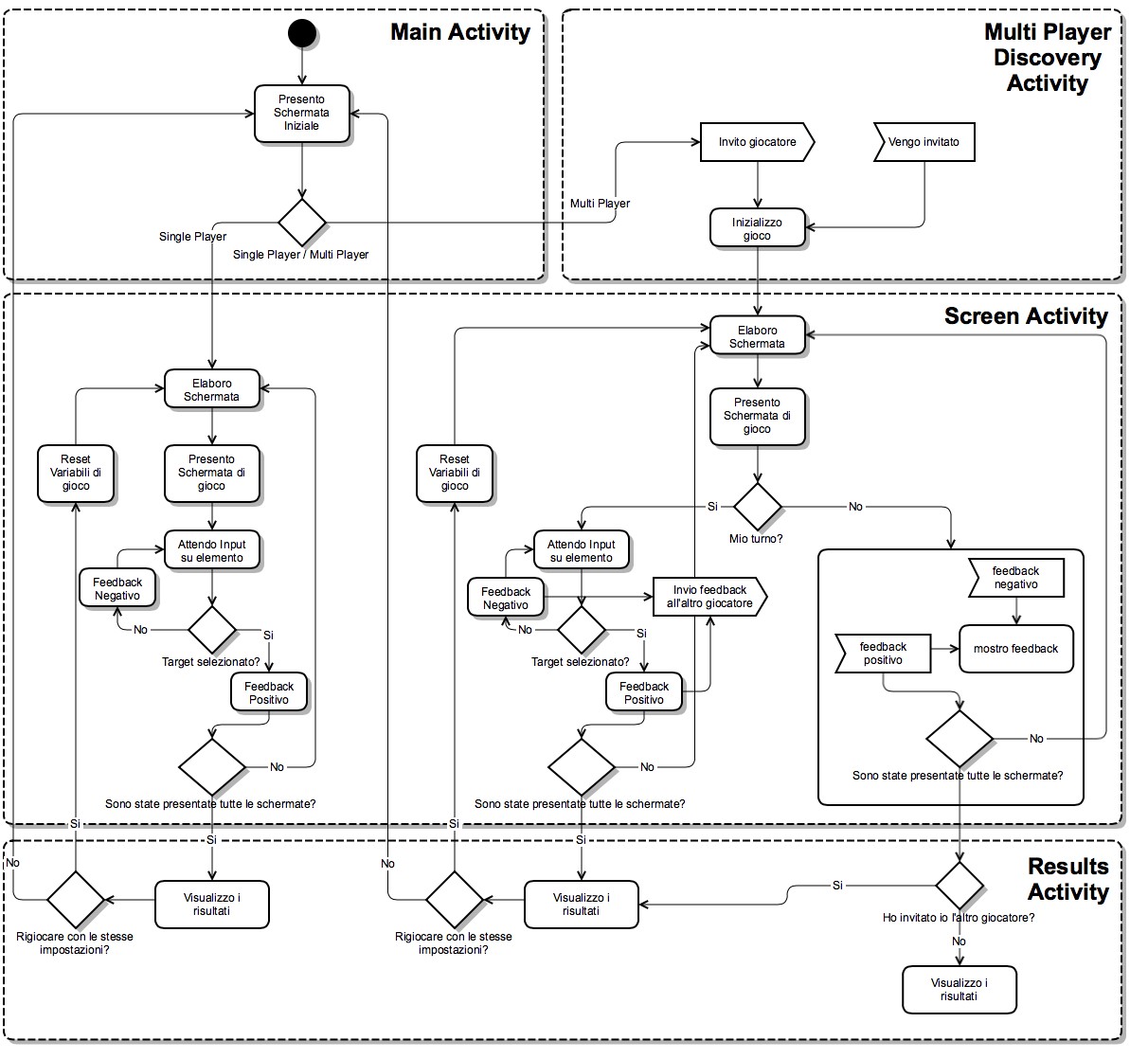


Figura 4.2: Class Diagram

**4.2 Manuale Utente**

Dalla schermata principale (figura 4.3), toccando l’icona grigia in alto a sini- stra dello schermo, si pu`o accedere alla schermata delle impostazioni (figura

4.4) da cui si possono impostare i parametri di gioco. Per impostare l’in- dirizzo e-mail per l’invio dei dati, invece, bisogner`a toccare “Configura il dispositivo” per accedere alla schermata in figura 4.5.



Figura 4.3: La schermata principale

Sempre dalla schermata iniziale si pu`o dare inizio al gioco sia in modalit`a multi-giocatore, che singolo. Se si sceglie la prima, allora si verr`a rimandati alla schermata di ricerca di altri giocatori connessi alla medesima rete *wi-fi* (figura 4.6).

Una volta iniziata la partita (figura 4.7), bisogner`a toccare gli intrusi per concluderla e vincere.



Figura 4.4: La schermata di impostazioni

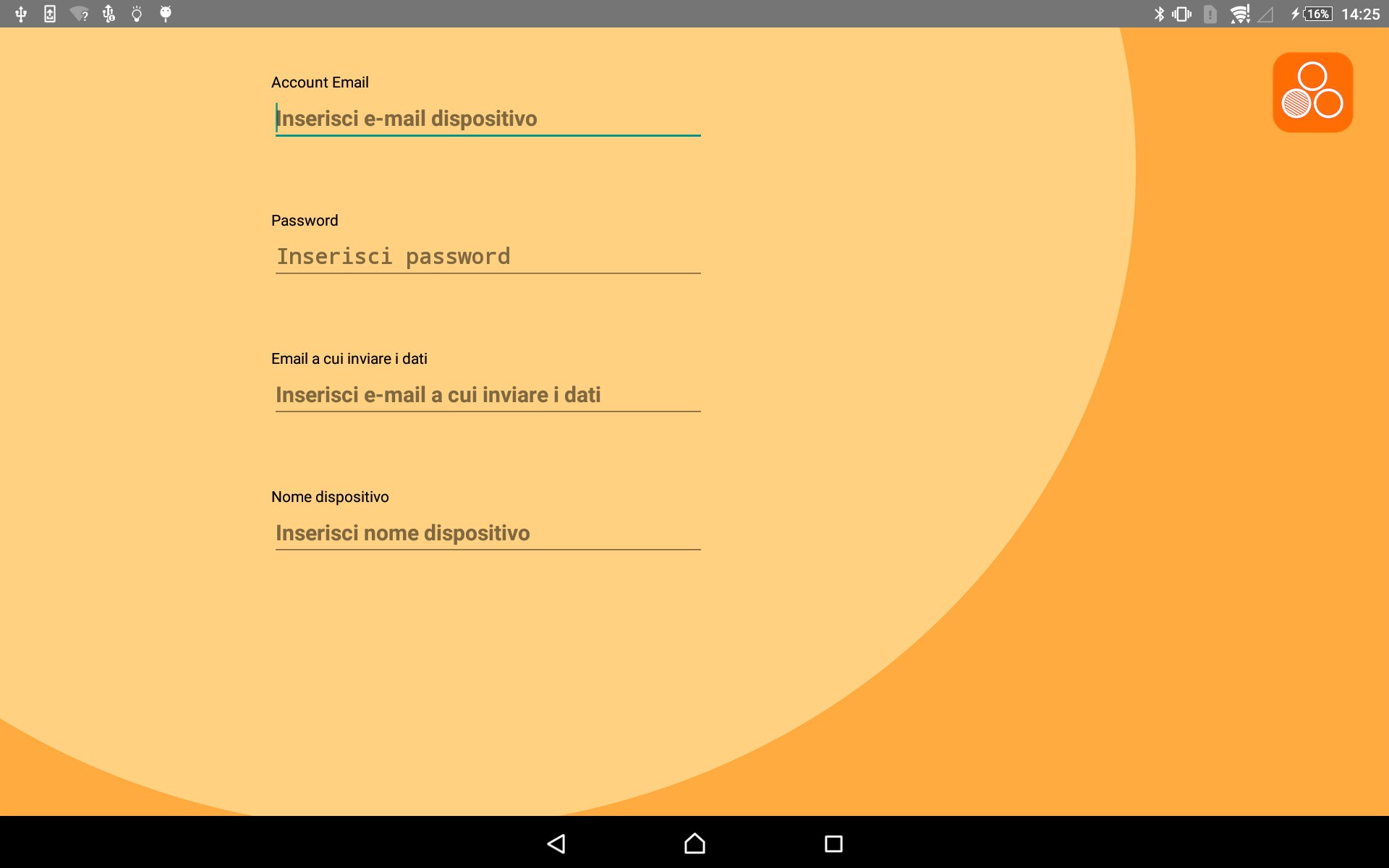


Figura 4.5: La schermata di impostazioni per l’invio dei dati di gioco

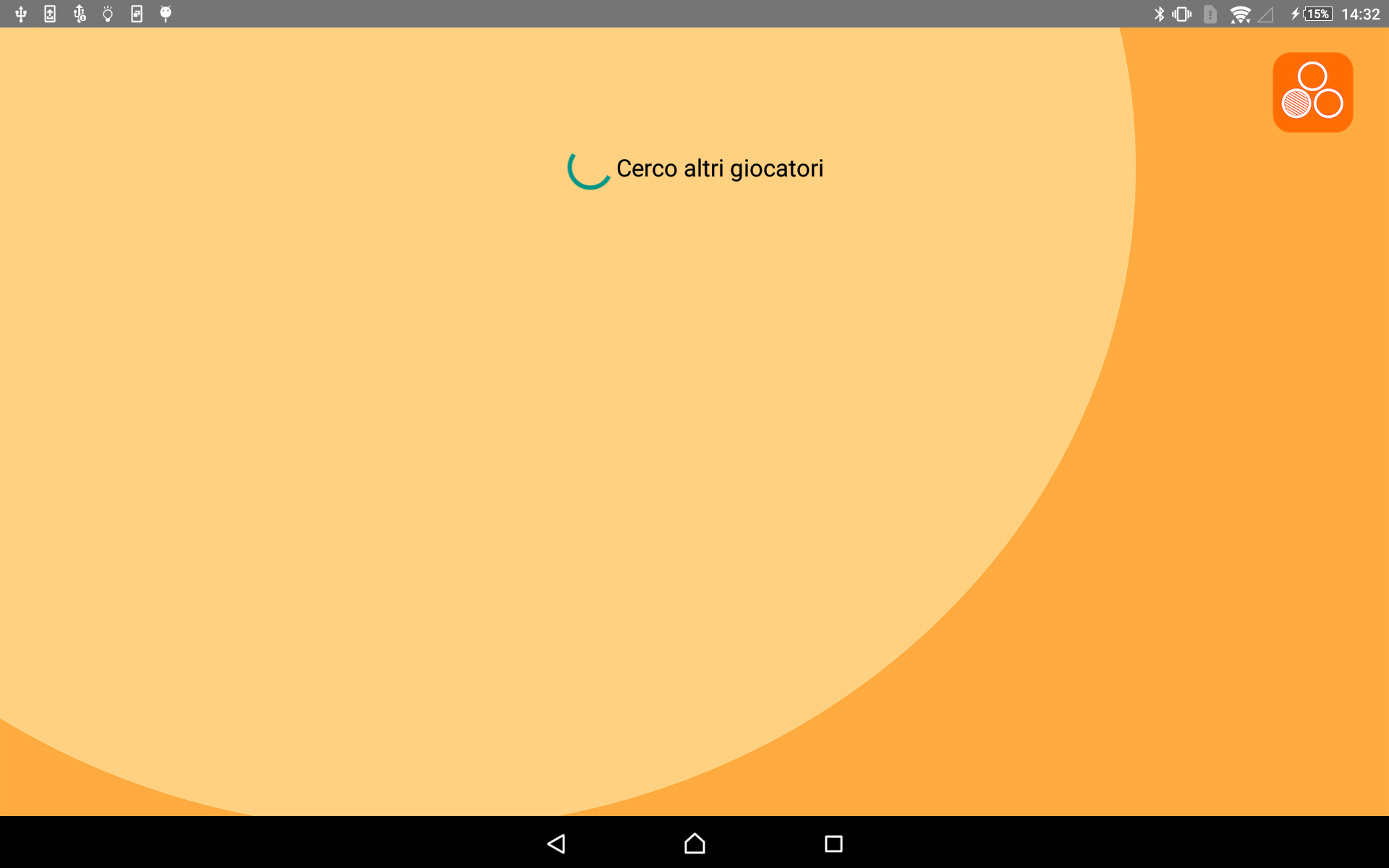


Figura 4.6: La schermata di ricerca di altri giocatori

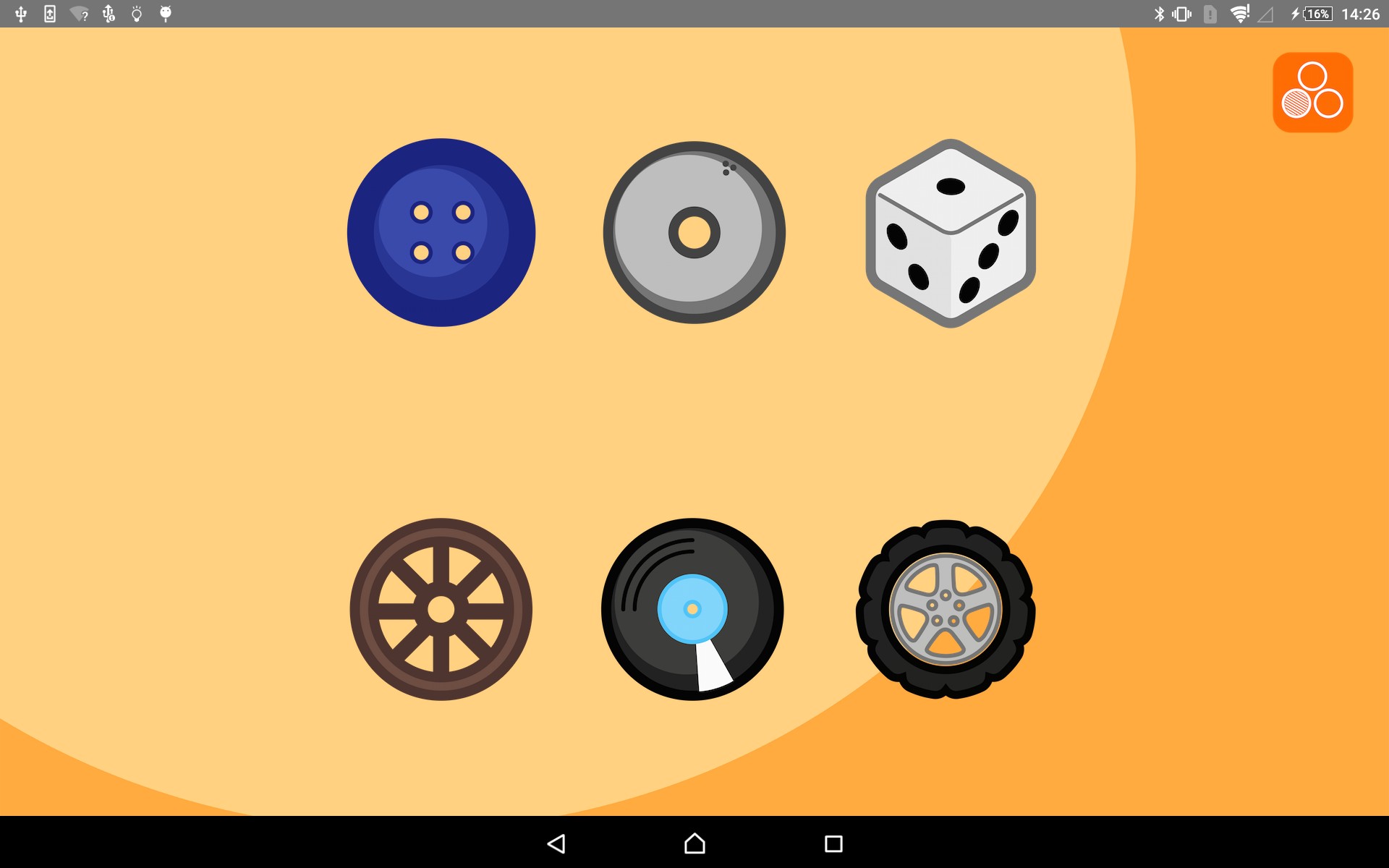


Figura 4.7: La schermata di gioco

**Capitolo 5**

**Trova l’Intruso 2**

*Documentazione a cura di Riccardo Medana* (r.medana@gmail.com)

**Indice**

5.1 Manuale Sviluppatore . . . . . . . . . . . . . . . . . . . . . . . 66

5.1.1 Descrizione delle classi . . . . . . . . . . . . . . . . . . 66

5.1.2 Strutture dati . . . . . . . . . . . . . . . . . . . . . . . 68

5.1.3 Modello della sessione di gioco . . . . . . . . . . . . . . 69

5.2 Manuale Utente . . . . . . . . . . . . . . . . . . . . . . . . . . 71

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5.2.1 | Primo avvio . . . . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 71 |
| 5.2.2 | Impostazioni di gioco | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 71 |

**5.1 Manuale Sviluppatore**

**5.1.1 Descrizione delle classi**

Vengono descritte di seguito le classi principali, iniziando da quelle che costi- tuiscono il modello dati. Successivamente vengono descritte le classi utility che permettono la gestione di elementi comuni alle varie parti dell’app, infine le Activity che compongono l’applicazione.

**Classi del modello dati**

Il modello dati del gioco `e rappresentato in figura 5.1.

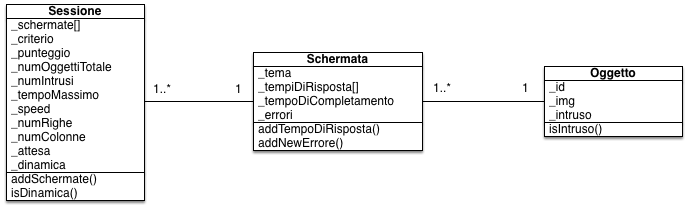


Figura 5.1: Class Diagram del modello dati

*•* **Oggetto** `e la classe che rappresenta il singolo oggetto sullo stage di gioco.

*•* **Schermata** `e la classe che rappresenta la singola schermata di gioco.

Una schermata contiene da 1 a N oggetti (N `e definito in base alla modalit`a di gioco)

*•* **Sessione** `e la classe che rappresenta l’intera sessione di gioco. Una sessione `e costituita da una o piu` schermate.

Tutte le classi sopracitate, oltre ai metodi particolari riportati nello sche- ma, espongono anche i relativi metodi getter e setter per accedere agli attri- buti.

**Classi utility**

*•* DbAdapter e DatabaseHelper sono due classi che contengono i metodi per la gestione del database delle email (le email da inviare al termine della partita vengono salvate in una tabella che agisce da coda di invio, per ogni email viene salvato il testo e l’indirizzo email di destinazione).

*•* GmailSender `e la classe che contiene la logica per poter inviare il re- port della sessione di gioco via mail. Questa classe utilizza la classe JSSEProvider per gestire l’autenticazione nella connessione al server di posta in uscita.

*•* ImgUtils contiene alcuni metodi per la gestione delle immagini all’in- terno del gioco

*•* JsonUtils contiene i metodi per accedere alle strutture dati JSON che definiscono le possibili schermate (v. Strutture Dati)

*•* MailUtils contiene il metodo che spedisce la mail in modalit`a asincro- na, assieme al metodo che compone il messaggio da spedire

*•* TimeUtils contiene il metodo che scrive il tempo in secondi in una stringa nel formato corretto.

*•* ViewContainer e ViewContainerAccessor sono due classi predefinite per la gestione delle animazioni con la libreria Universal Tween Engi- ne, definiscono il contenitore per la view da animare e il modo in cui dev’essere animata.

**Activity**

*•* MainActivity `e l’activity presentata all’avvio dell’app, propone le due modalit`a di gioco e il setup.

*•* SetupActivity `e l’activity che consente di modificare i parametri di gioco.

*•* ModeOneActivity `e l’activity che esegue la modalit`a di gioco dinamica

(con oggetti in movimento)

*•* ModeTwoActivity `e l’activity che esegue la modalit`a di gioco statica

(griglia con oggetti fissi nello spazio)

La descrizione dei singoli metodi contenuti nelle activity `e riportata al- l’interno del codice.

**5.1.2 Strutture dati**

All’interno della cartella Assets sono riportati alcuni file JSON che descri- vono le possibili schermate per ogni criterio (forma, percettivo, colore). Ad esempio, il file percettivo.json:

Listing 5.1: Contenuto del file percettivo.json

{

"scene" : [

{

"id" : "universo", "nome" : "Universo", "target" : "stella", "elementi" : [

{"nome" : "pianeta\_1"},

{"nome" : "pianeta\_2"},

{"nome" : "pianeta\_3"}

],

"sfondo" : "bg\_universo"

},

{

"id" : "prato", "nome" : "Prato",

"target" : "quadrifoglio", "elementi" : [

{"nome" : "trifoglio"}

],

"sfondo" : "bg\_prato"

},

{

"id" : "natura", "nome" : "Natura", "target" : "tartaruga", "elementi" : [

{"nome" : "anguria"}

],

"sfondo" : "bg\_prato"

}

]

}

Per ogni scena vengono specificati quali sono gli elementi grafici da uti- lizzare per comporla.

**5.1.3 Modello della sessione di gioco**

Nella figura 5.2 `e riportato il flusso di gioco, con riferimento alle attivit`a contenute nelle activity.

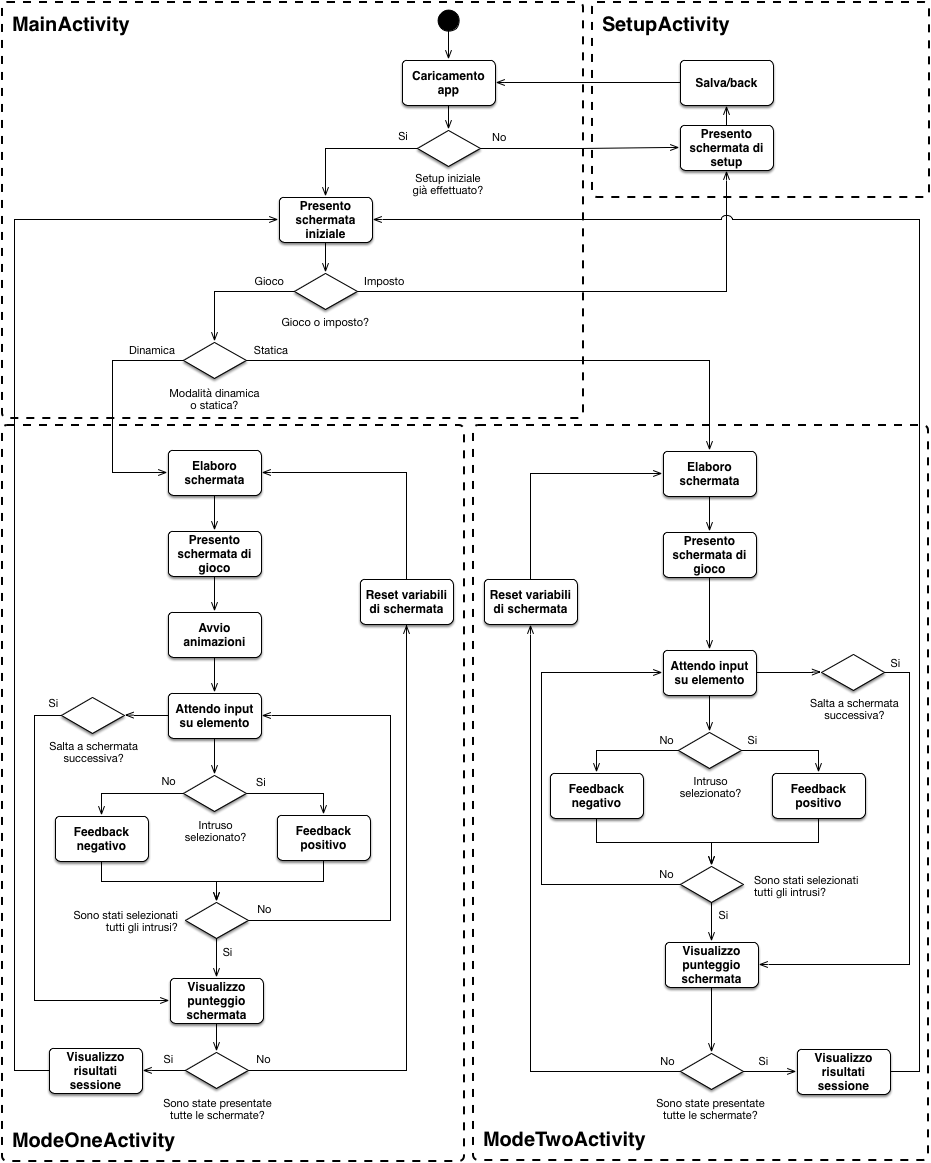


Figura 5.2: Modello della sessione di gioco

**5.2 Manuale Utente**

**5.2.1 Primo avvio**

Al primo avvio del gioco viene richiesto di effettuare il setup con i parametri necessari a giocare. Specificare quindi gli indirizzi email per l’invio e la ricezione dei report di gioco, assieme ai parametri di gioco. Premere “Salva impostazioni” per tornare alla schermata iniziale (si veda figura 5.3). Fino a quando non vengono salvate le impostazioni non `e possibile cominciare a giocare.

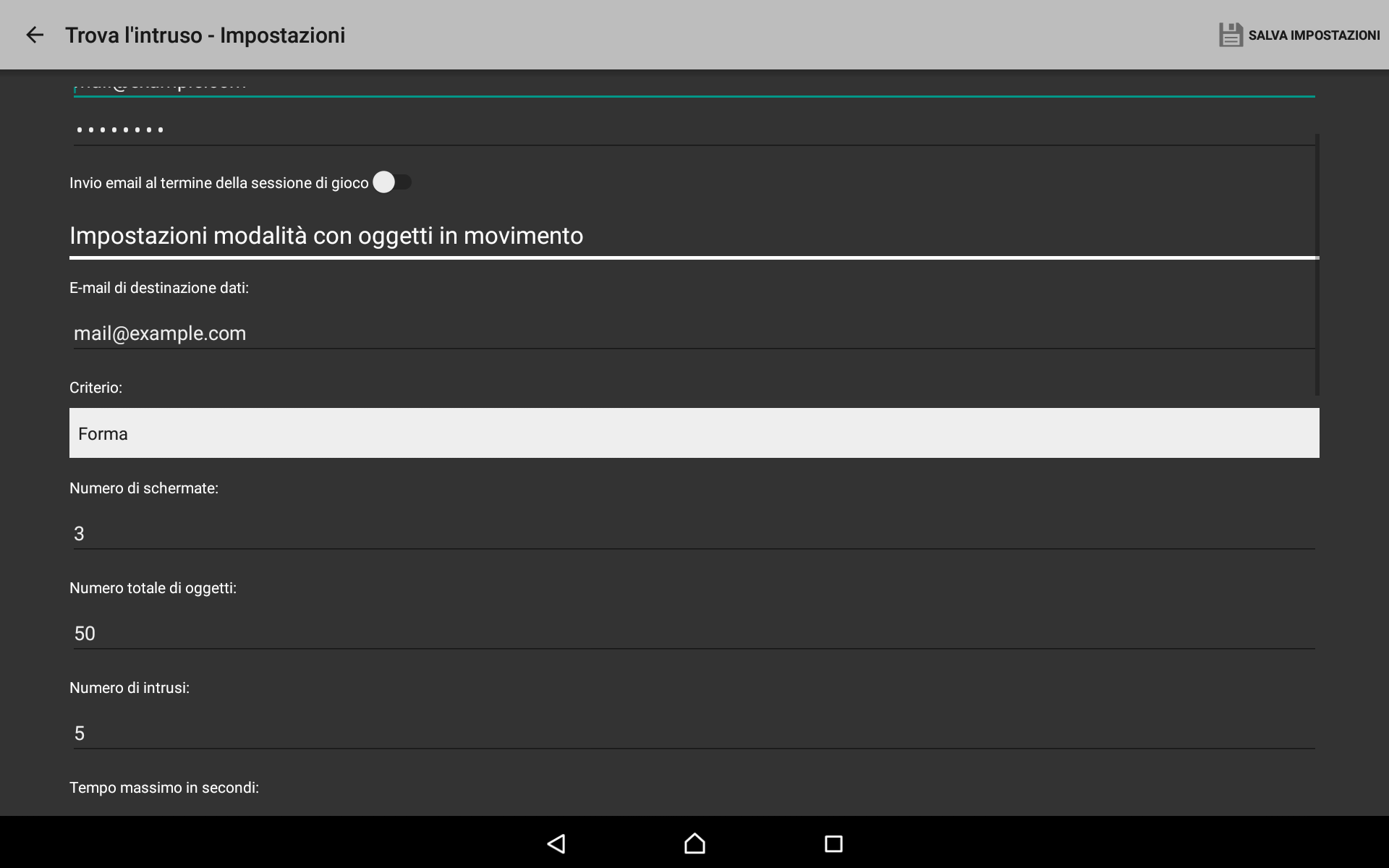


Figura 5.3: La schermata di impostazioni

**5.2.2 Impostazioni di gioco**

Per accedere alle impostazioni di gioco toccare l’icona grigia in alto a sinistra nella schermata iniziale (figura 5.4).



Figura 5.4: La schermata principale

**Impostazioni e-mail**

Per l’invio della mail `e necessario un account Gmail. Nel caso in cui si doves- sero riscontrare problemi con l’invio della mail `e necessario abilitare l’acces- so ad app meno sicure andando qui: [https://www.google.com/settings/ security/lesssecureapps](https://www.google.com/settings/security/lesssecureapps) I messaggi inviati dall’app saranno disponibili nella cartella Posta Inviata dell’account Gmail utilizzato. La mail viene in- viata solo se il selettore “Invio mail al termine della sessione di gioco” `e posizionato su ON, se `e attiva la connessione a Internet e solo se sono state visualizzate tutte le schermate (anche saltandole nel corso del gioco). La mail viene inviata dopo la comparsa della finestra “Partita terminata”, premendo sul pulsante di uscita. Nel caso in cui i dati di accesso a Gmail fossero errati, o non fosse presente una connessione a Internet attiva al termine della parti- ta, viene mostrato un messaggio di errore. La mail in questo caso non viene

persa, ma salvata in una coda di invio. Le mail da spedire vengono inviate al termine della prima partita in cui la connessione a Internet `e attiva e se i dati di accesso a Gmail sono corretti.

**Impostazioni modalit`a con oggetti in movimento**

In questa modalit`a il bambino deve individuare gli oggetti intrusi nascosti in un gruppo di oggetti in movimento (figura 5.5).

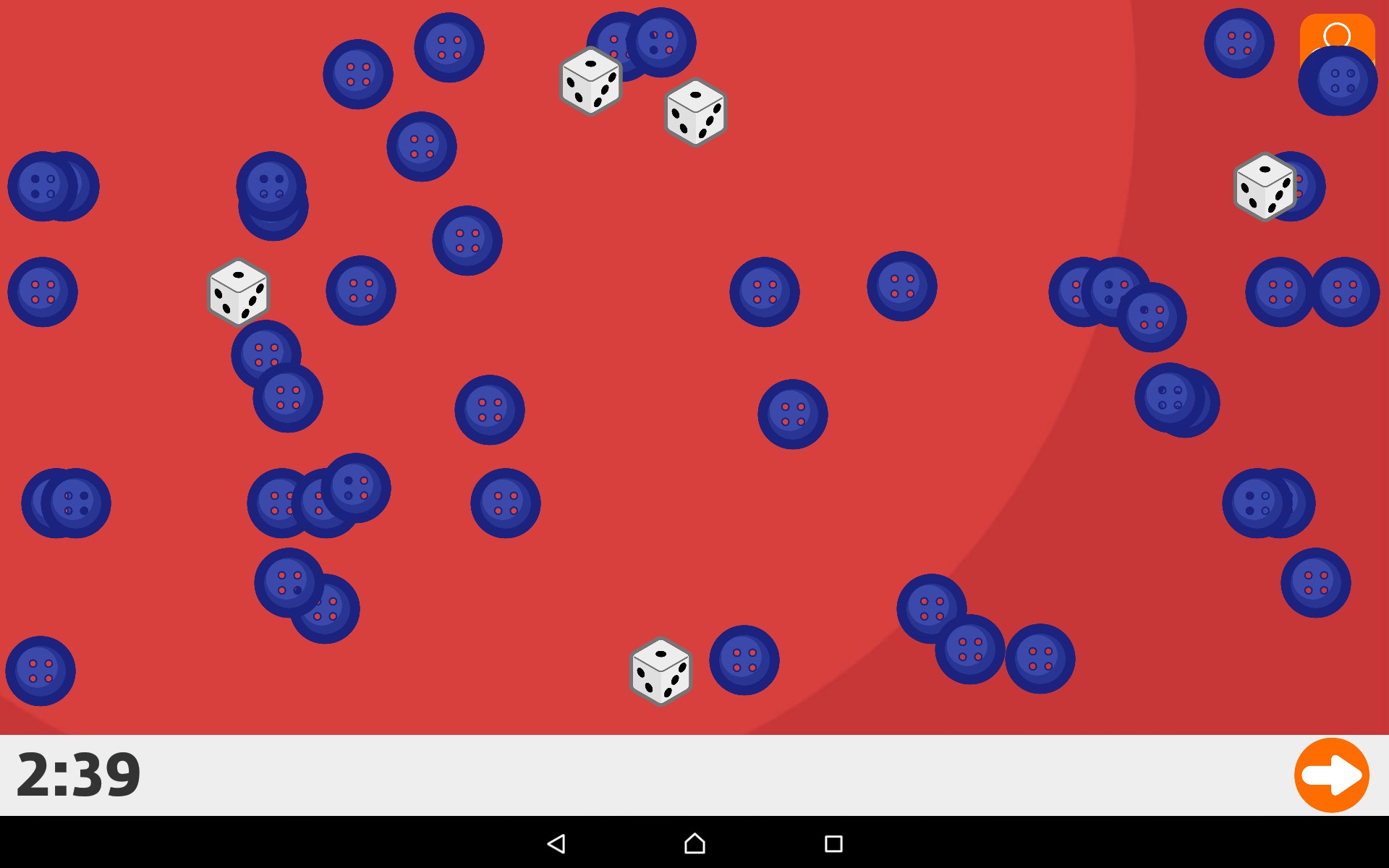


Figura 5.5: La modalit`a di gioco con oggetti in movimento

Occorre specificare:

*•* Email di destinazione dati (pu`o essere anche un indirizzo email non

Gmail)

*•* Criterio di gioco (forma/colore/percettivo)

*•* Numero di schermate da giocare

*•* Numero totale di oggetti sullo schermo (non piu` di 50, inclusi gli intrusi)

*•* Numero di intrusi nel gruppo di oggetti (in numero inferiore al totale degli oggetti)

*•* Tempo massimo di gioco per schermata (in secondi)

**Impostazioni modalit`a con oggetti fissi nello spazio**

In questa modalit`a il bambino deve individuare gli oggetti intrusi nascosti in una griglia di oggetti. Gli intrusi appaiono e scompaiono nel corso della partita, occorre toccarli prima che scompaiano (figura 5.6).



Figura 5.6: La modalit`a di gioco con oggetti fissi

Occorre specificare:

*•* Email di destinazione dati (pu`o essere anche un indirizzo email non

Gmail)

*•* Criterio di gioco (forma/colore/percettivo)

*•* Numero di schermate da giocare

*•* Numero di righe (non piu` di 6)

*•* Numero di colonne (non piu` di 10)

*•* Numero di intrusi (in numero inferiore al totale degli oggetti)

*•* Tempo massimo di gioco per schermata (in secondi)

*•* Tempo di attesa prima che compaia l’intruso (in secondi)

*•* Tempo di esposizione dell’intruso sullo schermo (in secondi)

**Capitolo 6**

**SuperApp**

*Documentazione a cura di Lorenzo Affetti* (lorenzo.affetti@polimi.it)

**Indice**

6.1 Manuale Sviluppatore . . . . . . . . . . . . . . . . . . . . . . . 78

6.2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6.1.1 | Introduzione . . . . | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 78 |
| 6.1.2 | Struttura Generale | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 79 |
| 6.1.3 | Backend . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 82 |
| 6.1.4 | Frontend . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 90 |
| Manuale Utente . . . . . . . . . . . . . . . . . . . . . . . . . . 94 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6.2.1 | *Master* . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 95 |
| 6.2.2 | *Slave* . . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 98 |

**6.1 Manuale Sviluppatore**

**6.1.1 Introduzione**

Lo scopo dell’applicazione quello di mettere in atto una collaborazione tra quattro giocatori attraverso diverse modalit`a di gioco. Ogni giocatore della squadra gioca su un tablet diverso, uno *slave*. Il gioco necessita di un ulteriore tablet, il *master*, che non verr`a utilizzato da nessun giocatore, ma che fornir`a informazioni di massima sulla partita in corso e scandir`a le partite e le manche di gioco.

I dispositivi comunicano per mezzo di una connessione *Bluetooth*. Un disegno di massima delle posizioni dei tablet `e fornito in figura 6.1.

L’applicazione comprende tre diversi giochi:

*• Trova l’Intruso*: I giocatori devono, appunto, trovare l’intruso. Ad un certo numero di risposte esatte consecutive (configurabile) la logica di gioco si inverte (vengono dati feedback sonori e visivi) e il gioco diventa, quindi, trova il *non* intruso. Questo gioco pu`o essere lanciato in quattro diverse modalit`a: trova l’intruso per colore, *ancora-colore* (come la modalit`a per colore, con la differenza che le immagini vengono mostrate in bianco e nero all’inversione di gioco), direzione e forma.

*• Ordina dal Piu`*

*Piccolo al Piu`*

*Grande* : I giocatori devono ordinare

quattro oggetti dal piu` piccolo al piu` grande o viceversa.

*• Costruisci la Torre* : I giocatori devono completare una torre composta da quattro oggetti giocando a turno.

In ognuno dei tre giochi l’applicazione sottopone agli utenti delle immagini che si muovono con velocit`a variabile su dei nastri trasportatori.

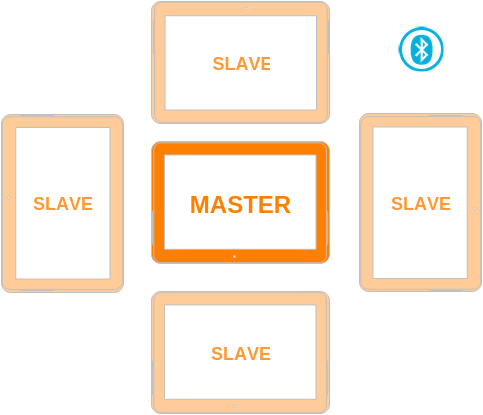


Figura 6.1: I cinque tablet

Ogni *partita* `e composta da un numero di *manche* (o *stage* ) configurabile. Il completamento di tutte le manche porta alla vittoria del gruppo. I punteggi sono sempre cumulativi dell’intera squadra.

**6.1.2 Struttura Generale**

Il codice sorgente dell’applicazione `e suddiviso in tre moduli (figura 6.2). Il modulo app contiene il backend, comprensivo di logica di gioco e di comu- nicazione, e il frontend dell’applicazione. Il modulo libgdx contiene le parti

piu`

interattive dell’applicazione e cio`e quelle dei nastri trasportatori. Infi-

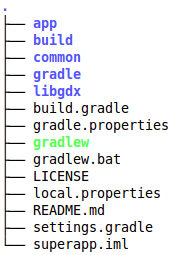


Figura 6.2: La struttura del progetto (profondit`a 1)

ne, il modulo common contiene moduli privi di dipendenze dal framework di

Android e usati in ambedue i precedenti moduli.

Come gi`a accennato, SuperApp `e divisa in tre diversi giochi di cui il pri- mo `e disponibile in quattro modalit`a diverse (in realt`a, anche il secondo, tuttavia la logica di gioco rimane invariata tra di esse, ci`o che varia `e so- lo l’ordinamento delle quattro immagini iniziali). Per questo motivo, quasi ogni classe costituente il nucleo della logica di gioco (si veda sezione 6.1.3) e diversi *fragment* (si veda sezione 6.1.4) segue il particolare schema di ere- ditariet`a illustrato in figura 6.3, a parte sporadiche eccezioni che, per`o, sono facilmente comprensibili a una prima lettura del codice (per esempio, abbia- mo Slave1Color e Slave1ColorAgain, ma solo un MasterColor poich´e il ruolo del *master* nelle due modalit`a non cambia).

Gli elementi costitutivi dell’intera applicazione sono:

**i *Controller*** rappresentano la logica di gioco e mantengono il suo stato;

**i *Tile*** o tesserini, costituiscono il modello dell’applicazione (si veda la classe

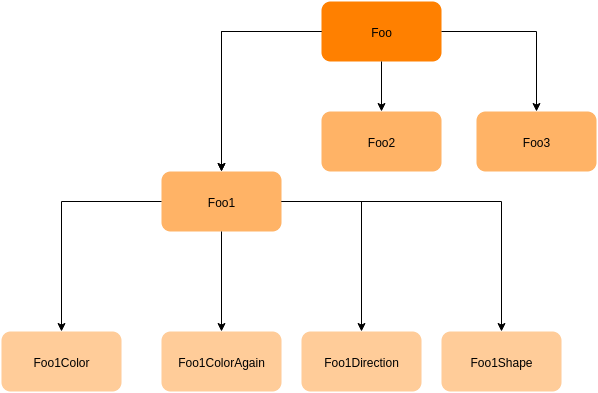


Figura 6.3: Lo schema di ereditariet`a generale

it.playfellas.superapp.tiles.Tile nel modulo common). Sono le immagini che l’utente toccher`a per dare delle risposte;

**la *UI*** mostra il gioco all’utente e raccoglie le sue azioni;

**i *Presenter*** interpretano le azioni dell’utente fornite dalla *UI* fornendole ai *Controller*. In base al responso ottenuto, vanno a informare la *UI* dei cambiamenti da apportare.

**il *Bus*** permette la comunicazione per mezzo di *eventi* tra gli elementi co- stitutivi dell’applicazione (in locale) e tra i dispositivi coinvolti (in remoto, tramite *Bluetooth* ). Per maggiori informazioni riguardo alla comunicazione, si veda la sezione 6.1.3.

Il diagramma in figura 6.4 mostra i vari componenti di SuperApp e la loro interazione ad alto livello su un dispositivo *slave*. Il diagramma fornito

non vuole essere esaustivo di tutti i possibili messaggi scambiati tra i mo- duli, bens`ı vuole chiarire le idee al lettore e fornire un’idea di massima delle interazioni tra le classi.

**6.1.3 Backend**

**Il TenBus**

Prima di spiegare il funzionamento di qualsiasi altro componente di Supe- rApp `e necessario introdurre il TenBus, parte del package it.playfellas. superapp.network. Questa classe `e quella che permette a tutte le compo- nenti del sistema di comunicare tra di loro – sia in remoto che in locale – per mezzo della trasmissione di *eventi*. Il TenBus `e un *wrapper* attorno al *bus* ad eventi **Otto** (<http://square.github.io/otto/>).

Dopo aver ottenuto un’istanza del TenBus tramite il metodo statico get, un oggetto pu`o postare NetEvent oppure InternalEvent. I primi verran- no inviati in remoto sul canale *Bluetooth*, i secondi verranno propagati in locale usando l’originale Otto. Gli eventi sono di molteplici tipi e le loro classi sono contenute nel package it.playfellas.superapp.events, la loro generazione `e accentrata nella EventFactory.

Se un oggetto volesse ricevere degli eventi, dovr`a semplicemente invocare il metodo register passando un qualsiasi oggetto (anche this) che abbia registrato dei metodi tramite l’annotazione @Subscribe. I methodi annotati devono accettare in ingresso un oggetto di tipo uguale all’evento interessato e ritornare un valore void.

Il TenBus offre anche i metodi attach e detach per permettere lo scam- bio di eventi in remoto. Per il funzionamento dettagliato della connessione *Bluetooth* (inizializzazione, chiusura) si veda il codice.

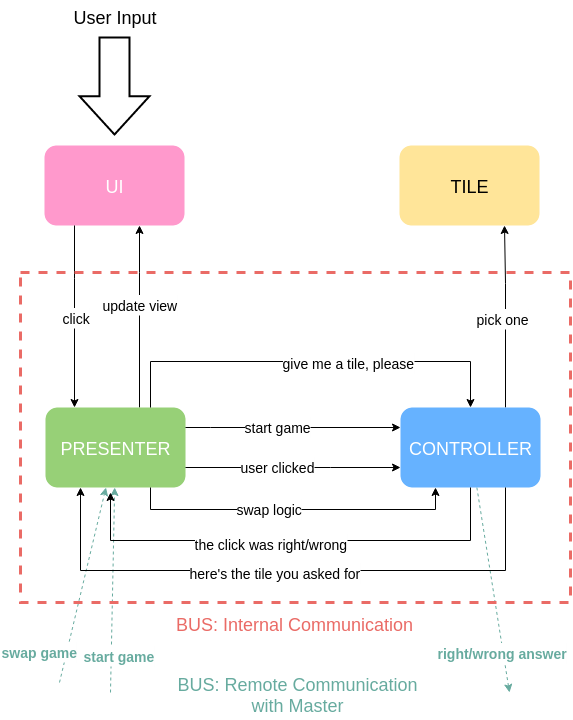


Figura 6.4: Le componenti di SuperApp su uno *slave*

Le classi BTThread e Peer (e quelle che le estendono) sono volutamente invisibili all’esterno del package network, l’unico punto d’accesso alla rete

per un oggetto esterno `e il TenBus.

**La Logica di Gioco**

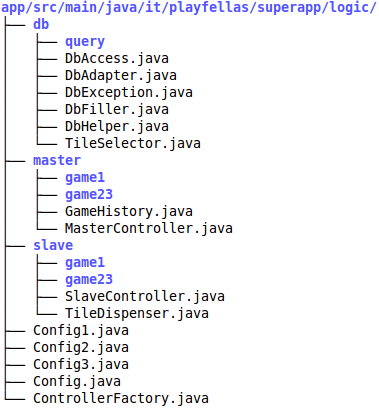


Figura 6.5: La struttura del package it.playfellas.superapp.logic

Gli elementi fondamentali della logica di gioco sono contenuti nel packa- ge it.playfellas.superapp.logic (per una struttura del package si veda figura 6.5):

**I Dispenser**

Sostanzialmente sono, come chiarisce la *signature* della classe it.playfellas. superapp.logic.slave.TileDispenser, degli Iterator<Tile>. Quel-

lo che un *dispenser* fa, infatti, `e fornire un nuovo *tile* ad ogni invoca- zione del metodo next. Ogni modalit`a di gioco ha un suo particolare

*dispenser* (il gioco 1 ne ha, invece, due: uno normale e l’altro per l’in- versione di gioco), il quale interroga il *database* dell’applicazione (che contiene solo *tile* ) per ottenere un insieme di tesserini. Questi ultimi verranno forniti ai *controller* tramite il metodo next secondo logiche proprie del *dispenser* stesso. Per indagare le logiche specifiche si veda il codice sorgente degli specifici TileDispenser contenuti nei package it.playfella.superapp.logic.slave.game1 e game23;

**I Master Controller**

I *controller* rappresentano la logica di logico. Ogni azione che l’utente compie per mezzo di un gesto sullo schermo del tablet viene interpretata dalla parte di *UI* e viene passata ad un *controller* che sapr`a interpre- tarla e fornire una risposta in base allo stato attuale della partita. I *master*, nello specifico, sono quelli che mantengono lo stato globale della partita (per esempio: il punteggio, i giocatori, ecc.) e danno inizio alle partite sugli *slave* (si veda il metodo beginStage) fornendo, a volte, informazioni condivise (per esempio, nel gioco 1, il colore/forma/dire- zione base; nei giochi 2 e 3, le quattro *tile* di base). Oltre a dare inizio alle partite, ne scandiscono ogni fase, come il loro inizio e la loro fine e l’inizio e la fine delle manche che le compongono.

**Gli Slave Controller**

Gli *slave* attendono che i *master* trasmettano gli eventi di inizio/fine partita e inizio/fine manche per poter dare inizio al gioco vero e pro- prio. Essi istanziano i *dispenser* per ottenere i prossimi *tile* da fornire agli utenti e contengono anche la parte di logica di gioco riguardante l’esattezza o meno di un *tile* date le sue caratteristiche (vedi metodo isTileRight). Quest’ultima informazione `e quella che verr`a trasmessa

tramite eventi al *master* che decider`a, di conseguenza, se incrementare o azzerare (o altro) il punteggio complessivo. Ogni SlaveController fornisce anche il metodo nextTile, il quale *delega* ad un *dispenser* l’ot- tenimento del prossimo *tile* (metodo next) da mostrare all’utente. Sar`a poi la parte di *UI* che mostrer`a il tesserino all’utente posizionandolo nel momento giusto sul nastro trasportatore usando il DisposingService (si veda it.playfellas.superapp.ui.slave).

Come precedentemente detto, la partita `e articolata in un numero confi- gurable di stage. Ognuno di essi prevede un punteggio massimo configurabile. Al raggiungimento di tale punteggio, lo stage termina e si d`a inizio allo stage successivo. Quando tutti gli stage sono stati completati, la partita finisce.

La scansione della partita nelle suddette fasi `e determinata dal MasterController. Esso invia, tramite il TenBus, un evento StartGameEvent (in realt`a, viene

inviato un evento che estende quella classe. Per esempio: StartGame1Color, oppure StartGame2Event, ecc.). Alla ricezione dell’evento, ogni *slave* reagi- sce mostrando la schermata di gioco all’utente. A questo punto, il *master* pro- cede per stage fino al loro esaurimento: viene inviato un BeginStageEvent e lo stage ha inizio; quando il punteggio massimo viene raggiunto, il *master* invia un EndStageEvent seguito da un ulteriore BeginStageEvent in caso vi sia un altro stage da giocare, altrimenti viene inviato un EndGameEvent.

In figura 6.6, viene rappresentata la successione di eventi che regolano lo svolgimento della partita.

**L’invio dei Dati di Gioco**

Per l’invio dei dati abbiamo utilizzato il *database* non relazionale di **Firebase**

(<https://www.firebase.com/docs/android/quickstart.html>).

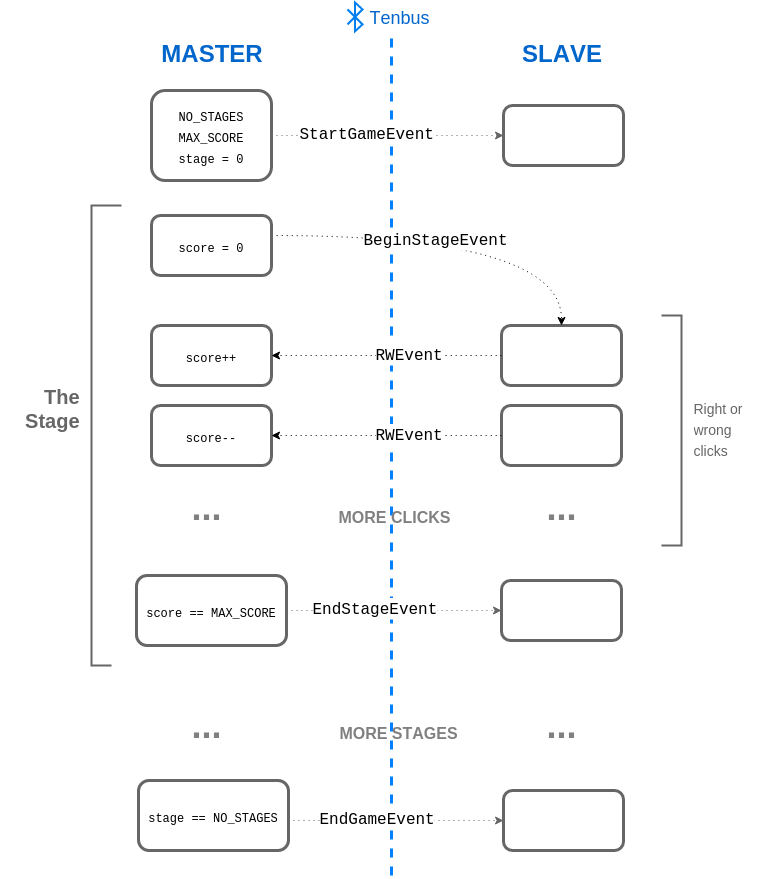


Figura 6.6: La partita

La classe GameHistory si prende carico di memorizzare tutte le azio- ni intraprese dai giocatori e il loro esito, tramite l’esposizione dei metodi right e wrong. Sar`a il MasterController a invocare questi metodi della GameHistory dopo aver ricevuto i RWEvent da parte degli *slave*. Alla fine

della partita, il *master* persiste i dati sul *backend Firebase*.

La disponibilit`a di connettivit`a internet non `e comunque necessaria pro- prio grazie al funzionamento interno di *Firebase* : i dati verranno infatti sal- vati nel suo *database* locale fino a che la connettivit`a sar`a disponibile, in quel momento, i dati verranno salvati nel *database* remoto.

La struttura dai dati `e quella di un oggetto *JSON*. I dati in remoto, infatti, possono essere esportati come file .json in qualsiasi momento.

Il dato di una partita contiene il suo *ID* (visibile sulla schermata di gioco

del *master* ), la sua storia completa (ogni singolo *click* da parte dei giocatori e l’esattezza o meno della risposta data) e piu` di dieci indici. La struttura del dato `e ancora da considerarsi variabile fino a sperimentazione conclusa.

Riportiamo un esempio di dato per una partita, troncato dove la struttura

dei dati `e ripetitiva e dove l’informazione `e ridondante, nel listato 6.1.

Listing 6.1: Un esempio di dato per una partita

{

"dd42247e" : { "history" : [ {

"deltaT" : 0.0, "player" : "player4", "rw" : false,

"ts" : "28/09/2015 13:34:06:074"

}, {

"deltaT" : 0.001, "player" : "player3", "rw" : true,

"ts" : "28/09/2015 13:34:06:075"

},

// more and more clicks ...

],

"index10\_noRightPerPlayer" : {

|  |  |
| --- | --- |
| "player\_player1" : | 10, |
| "player\_player2" : | 7, |
| "player\_player3" : | 11, |

"player\_player4" : 11

},

"index11\_noWrongPerPlayerPerStage" : { "player\_player1" : {

"stage\_1" : 8, "stage\_2" : 3, "stage\_3" : 3, "stage\_4" : 1

},

// more players ...

},

"index12\_noWrongPerPlayer" : {

// hidden

},

"index13\_ratio9\_11" : {

// hidden

},

"index14\_ratio10\_12" : {

// hidden

},

"index15\_clicksRatio" : {

// hidden

},

"index1\_elapsedTime" : 9.166666666666666E-4, // very low because it was auto-generated

"index2\_noWrongPerStage" : { "stage\_1" : 24,

"stage\_2" : 12, "stage\_3" : 13, "stage\_4" : 12

},

"index3\_noWrong" : 61, "index4\_noRightPerStage" : {

// hidden

},

"index5\_noRight" : 39, "index6\_playerContributionPerStage" : {

"player\_player1" : { "stage\_1" : {

"1-3" : 0.23076923076923078, "2-3" : 0.46153846153846156,

"3-3" : 0.35714285714285715

},

// more stages...

},

// more players...

},

"index7\_balancePerStage" : { "stage\_1" : {

"1-3" : 5.25, "2-3" : 7.25, "3-3" : 3.5

},

// more stages...

},

"index8\_playerContributionStabilityPerStage" : {

// hidden

},

"index9\_noRightPerPlayerPerStage" : {

// hidden

}

}

}

I dati sono salvati all’indirizzo <https://giocoso2015.firebaseio.com/>. Per le credenziali di accesso, si contatti [lorenzo.affetti@polimi.it.](mailto:lorenzo.affetti@polimi.it)

**6.1.4 Frontend**

**Le Activity e i Fragment**

In questa sezione riportiamo un elenco delle *activity* e dei *fragment* e delle loro principali mansioni. Si noti che, a fianco dei nomi di *activity* o *fragment*,

`e riportato un riferimento ad una figura della sezione 6.2 rappresentante la corrispettiva visuallizzazione sul dispositivo; ci`o `e stato fatto per garantire una maggiore chiarezza verso il lettore/sviluppatore che, in caso di manuten-

zione futura, sapr`a dove “mettere le mani” in caso di errori e/o aggiunta di funzionalit`a.

Per il codice sorgente si faccia riferimento al package it.playfellas. superapp.ui.

**MainActivity (fig. 6.10)**

`e il punto di accesso dell’applicazione, permette di scegliere tra *master*

e *slave* ;

**master.MasterActivity (fig. 6.12)**

permette la scelta tra i diversi giochi (1, 2 e 3);

**master.GameActivity**

`e l’*activity* in cui avviene la partita nel *master*. Contiene i seguenti

*fragment* :

**SettingsFragment (fig. 6.13)**

permette di configurare il gioco corrente;

**GameFragment (fig. 6.14)**

mostra lo stato della partita (stage, punteggio, foto dei gioca- tori, ecc.). Istanzia il giusto (per la modalit`a di gioco corrente) GamePresenter che a sua volta istanzia il giusto MasterController.

**master.bluetooth.BluetoothActivity (fig. 6.11)**

permette l’associazione dei dispositivi.

**master.bluetooth.FastStartActivity (fig. 6.15)**

permette l’avvio rapido del *master* riassociando gli ultimi dispositivi connessi.

**slave.SlaveActivity (fig. 6.16)**

*activity* in cui lo *slave* aspetta di venir scelto dal *master* come giocatore della partita tramite associazione *Bluetooth* ;

**slave.GameActivity**

`e l’*activity* in cui avviene la partita nello *slave*. Tramite la ricezione degli eventi che estendono StartGameEvent, determina la modalit`a di gioco corrente e sceglie, di conseguenza, il giusto *fragment* da mostrare. Contiene i seguenti *fragment* :

**PhotoFragment (fig. 6.17)**

l’utente pu`o scattarsi una foto da utilizzare come *avatar* durante la partita;

**SlaveGameFragment (fig. 6.18, 6.19 e 6.20)**

contiene i nastri trasportatori e i *tiles* (vedi sezione 6.1.4). Istanzia i nastri trasportatori e il giusto SlavePresenter (che istanzia il giusto SlaveController) in base alla modalit`a di gioco corrente.

Per l’istanziazione dei *controller* si faccia riferimento a it.playfellas. superapp.logic.ControllerFactory.

**I Nastri**

Il modulo del progetto che gestisce la visualizzazione e lo scorrimento dei na- stri e la rappresentazione e animazione dei *tile* `e libgdx. **LibGDX** ([https:](https://libgdx.badlogicgames.com/features.html)

[//libgdx.badlogicgames.com/features.html](https://libgdx.badlogicgames.com/features.html)) `e un framework Java che

utilizza **OpenGL** (<https://www.opengl.org/>) per la renderizzazione vi- deo. Il risultato della scelta di questo framework `e un miglioramento sia nella fluidit`a delle animazioni, che nell’occupazione della memoria del di-

spositivo da parte dell’applicazione rispetto all’utilizzo dell’ObjectAnimator

standard di Android.

*LibGDX* racchiude i concetti piu` comuni dei framework di renderizzazione grafica. Troviamo infatti la *scena*, il “luogo”in cui avvengono le animazioni, e gli *sprite*, immagini bidimensionali dotate di posizione, *texture* (la loro immagine), altezza e larghezza.

Gli elementi costitutivi del modulo libgdx sono:

**la Scene** la *scena* del gioco;

**i Conveyor** i nastri trasportatori su cui vengono posati i *tile*. Essi possono essere di diversi tipi. Troviamo infatti MovingConveyor (figura 6.7), SizeConveyor (figura 6.8) e TowerConveyor (figura 6.9).

Bisogna notare che gli ultimi due non sono propriamente nastri tra- sportatori in quanto sono statici. Essi vengono utilizzati nei giochi 2 e

3 rispettivamente.

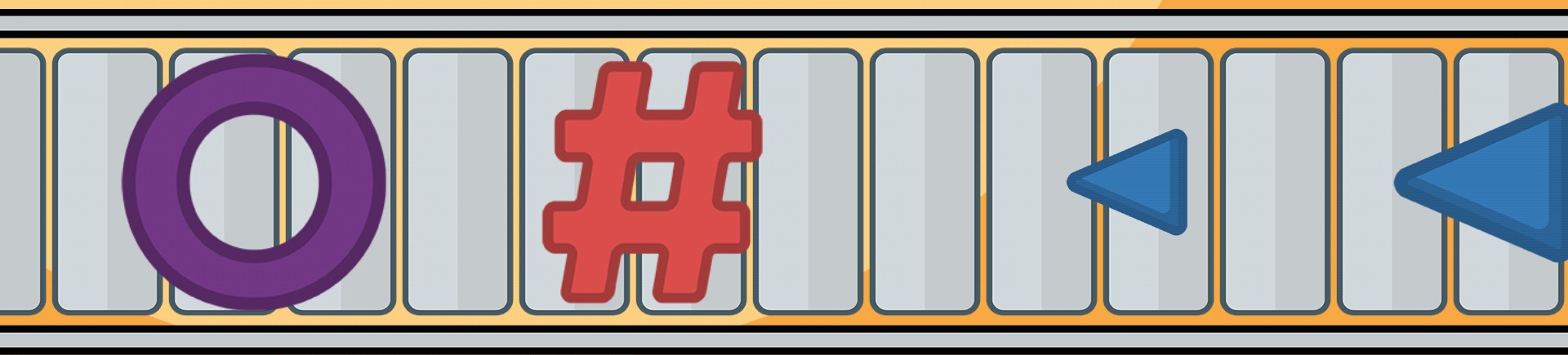


Figura 6.7: Un nastro trasportatore

**le TileRepr** dei *wrapper* intorno ad un Tile e un SimpleSprite. Essi sono la rappresentazione concreta di un *Tile* (quello del modello del- l’applicazione). La corrispondenza TileRepr (rappresentazione), Tile (modello) `e talmente stretta e semanticamente equivalente che, da ora in poi, utilizzeremo i due termini in modo intercambiabile.

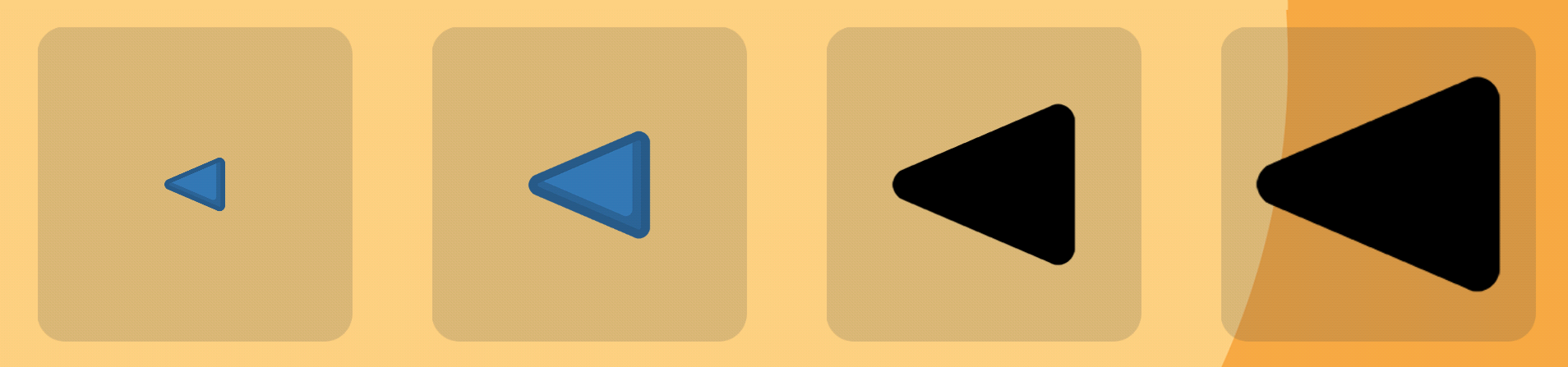


Figura 6.8: Il “nastro trasportatore”statico del gioco 2



Figura 6.9: Il “nastro trasportatore”statico del gioco 3

Ci`o che avviene ad ogni frame di visualizzazione `e l’esecuzione del metodo render della Scene, il quale aggiunge ogni *sprite* presente nella scena ad un *batch* di renderizzazione, dopo averne aggiornato posizioni, texture e dimen- sioni. Il *batch* cos`ı generato, contenente una matrice di pixel che rappresenta il “disegno”degli *sprite* aggiunti, viene renderizzato tramite l’invocazione di batch.end().

Per comprendere cosa si intende per “aggiornamento”di uno *sprite*, si pen-

si alla sua posizione: per far muovere un nastro o un *tile*, infatti, `e necessario cambiare la sua posizione pixel per pixel ad ogni frame di rappresentazione (si vedano i metodi update nelle classi che estendono Conveyor).

**6.2 Manuale Utente**

Al primo avvio dell’applicazione viene mostrata la schermata principale (fi- gura 6.10) dalla quale `e possibile scegliere tra *master* e *slave*.

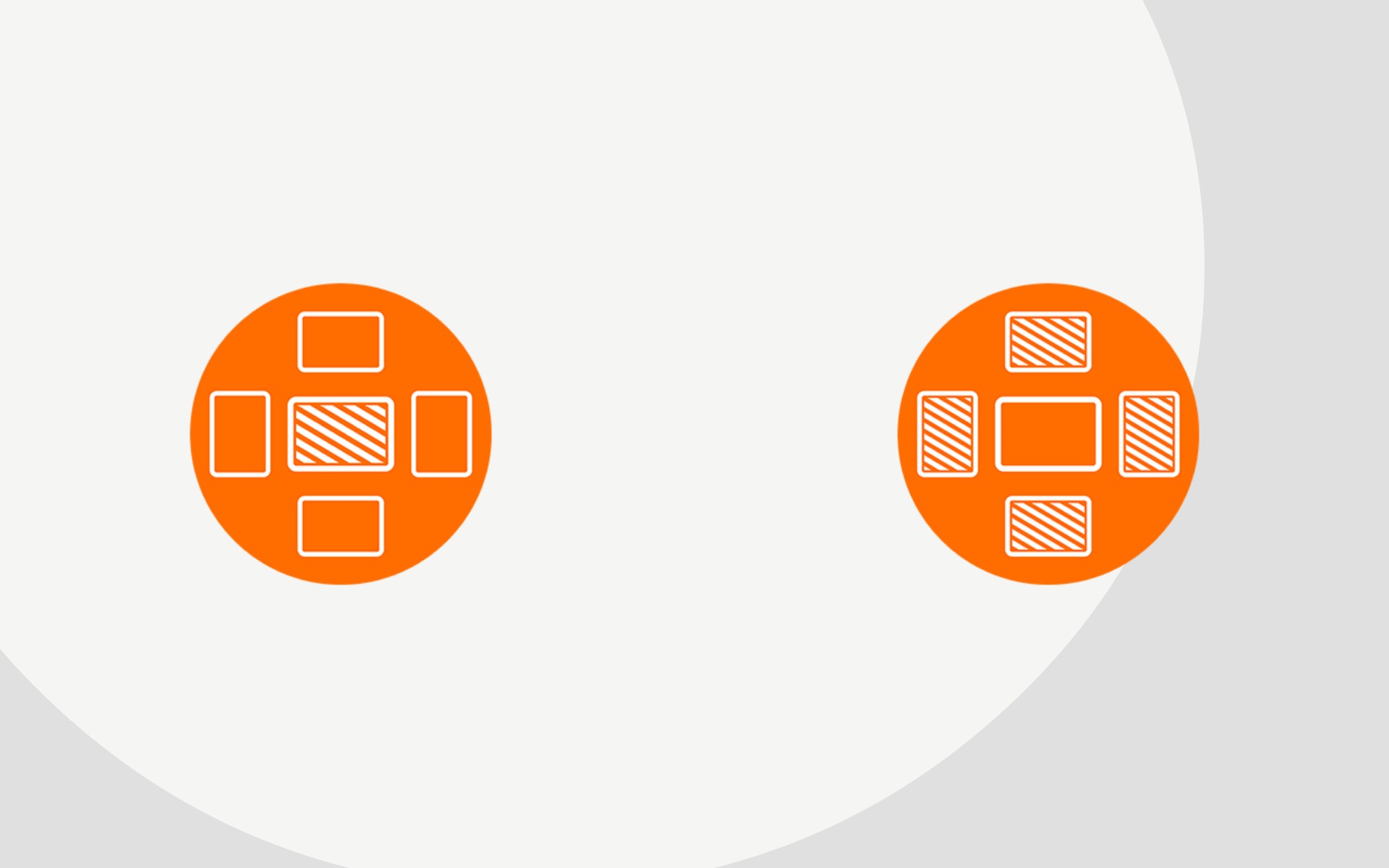


Figura 6.10: La schermata principale

La scelta determina sostanzialmente il funzionamento dell’applicazione: per l’applicazione *master* si faccia riferimento alla sotto-sezione 6.2.1, altri- menti alla sotto-sezione 6.2.2.

**6.2.1 *Master***

Una volta scelto di essere *master*, bisogner`a scegliere gli *slave* che prenderan- no parte alla prossima partita. Questa operazione viene effettuata tramite connessione *Bluetooth* come nella schermata in figura 6.11. Soprattutto per avere coerenza nel meccanismo a turni del terzo gioco, l’applicazione chiede all’utente di inserire i dispositivi uno ad uno in senso orario indicando quale deve essere il prossimo ad essere aggiunto evidenziando il corrispondente lato dello schermo in arancione, fino ad un massimo di 4 dispositivi. Il processo

`e effettuato “per lato” poich´e l’applicazione `e progettata per rispondere ad

una configurazione topologica dei tablet come in figura 6.1.

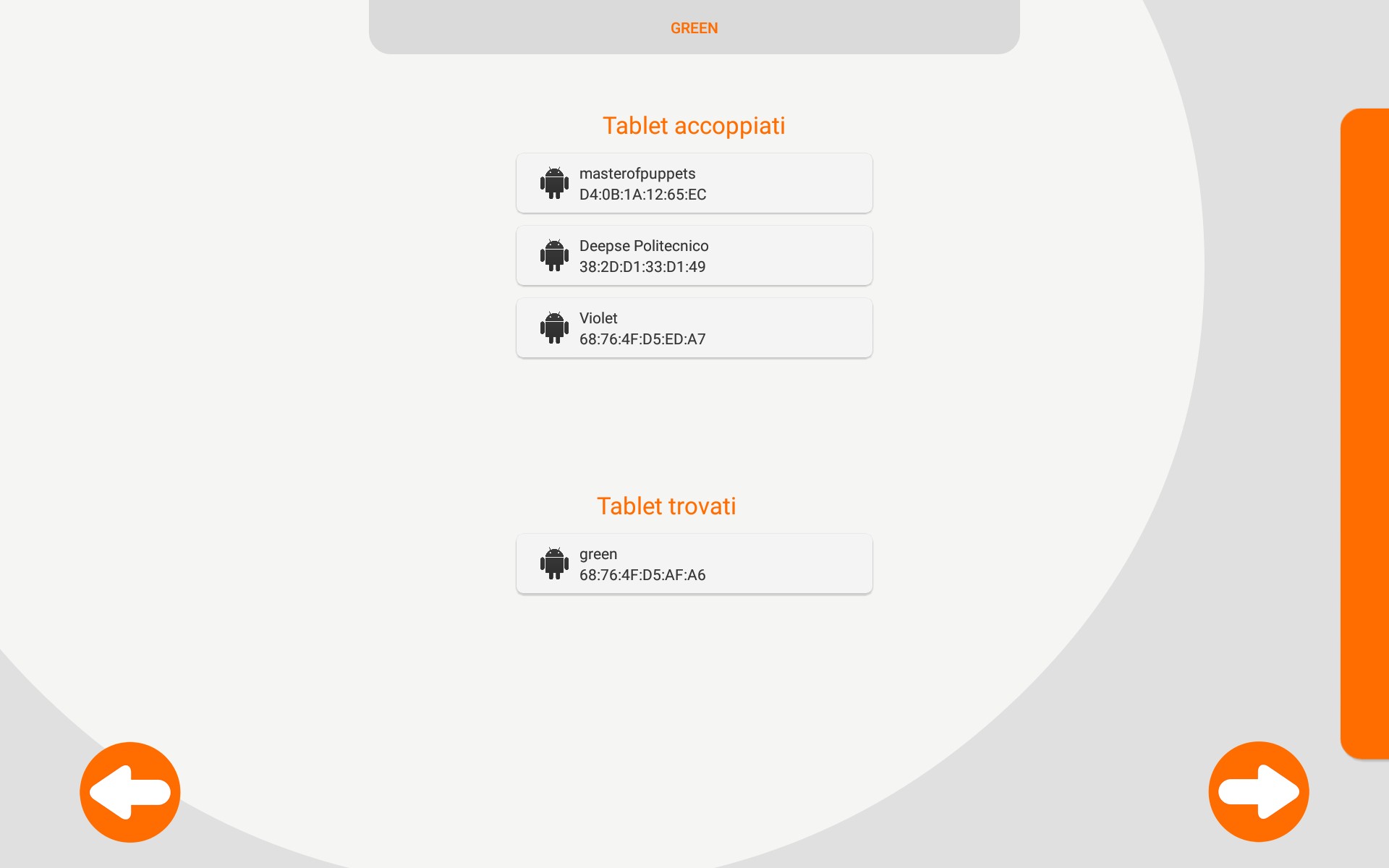


Figura 6.11: La connessione *master* /*slave* tramite *Bluetooth*

Una volta scelti i giocatori, si potr`a scegliere uno dei tre possibili giochi

(figura 6.12) e, successivamente, impostare i parametri della partita (figura

6.13).

Una volta iniziata la partita, premendo la freccia nel cerchio arancione in basso a destra, viene mostrata la schermata di stato (figura 6.14) contenente l’immagine del capitano, che `e tanto colorata quante *manche* di gioco sono state completate. Cliccando sul piccolo pulsante grigio in alto a destra con- tenete un punto interrogativo, si pu`o conoscere lo stato attuale nel dettaglio ed interrompere forzatamente la partita in corso.

In caso il *master* abbia gi`a effettuato una volta la configurazione dei gio- catori della partita, l’applicazione mostrer`a la schermata di configurazione automatica (figura 6.15) all’avvio. L’applicazione tenter`a, infatti, di rieffet-



Figura 6.12: La selezione del gioco

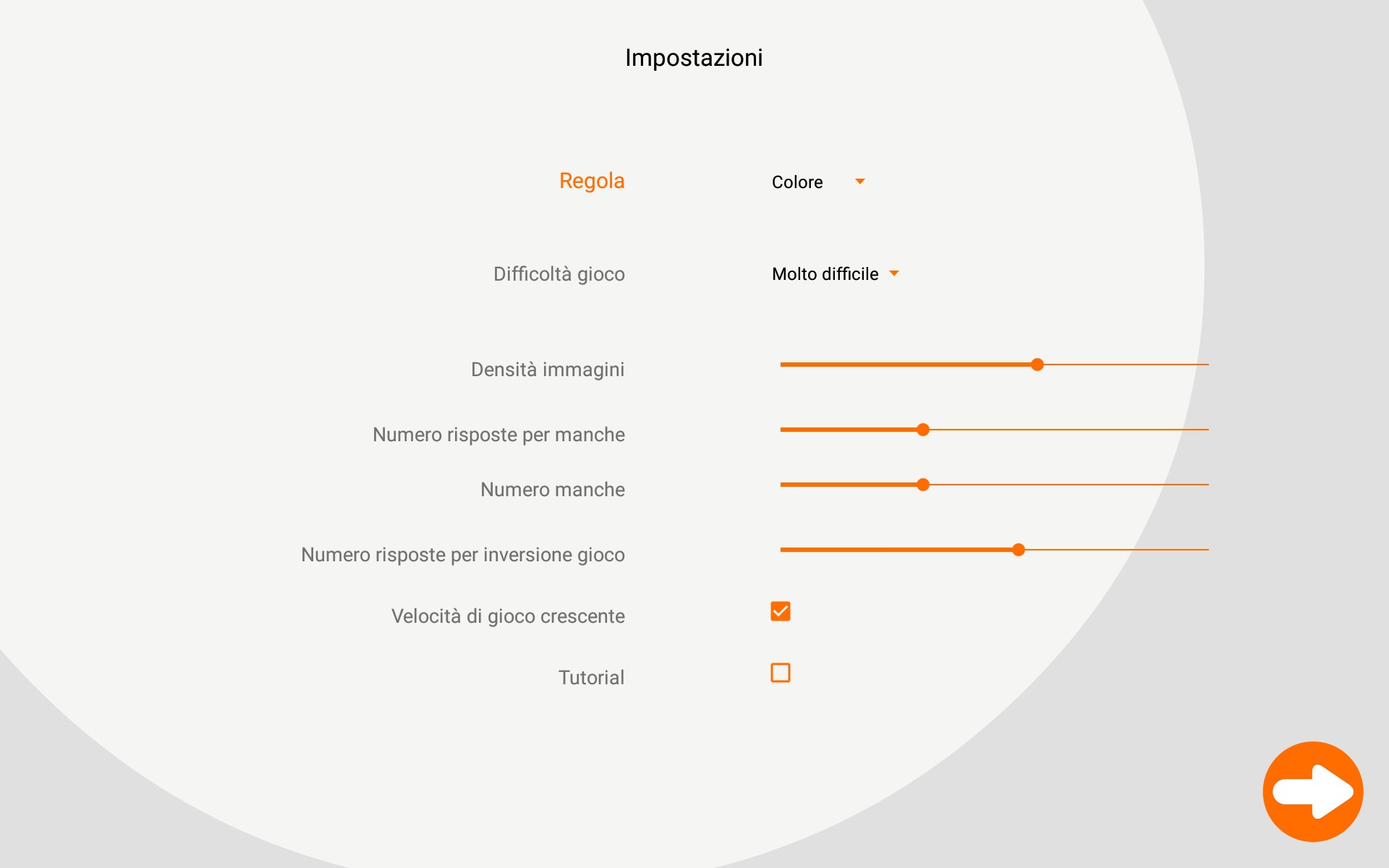


Figura 6.13: Il *master* configura la partita che sta per iniziare



Figura 6.14: La schermata che mostra lo stato della partita sul *master*

tuare le connessioni come gi`a erano state configurate. Questa operazione pu`o comunque essere interrotta dando luogo ad una nuova fase di configurazione.

**6.2.2 *Slave***

Lo *slave* attende che il *master* si connetta a lui (figura 6.16).

Una volta che la connessione `e stata effettuata, viene data la possibi- lit`a di scattare una foto (figura 6.17) che verr`a usata come *avatar* di gioco, nell’attesa dell’inizio della partita.

Una volta che il *master* avr`a configurato i parametri e avr`a dato inizio alla partita, allora gli *slave* potranno giocare fino alla vittoria (figure 6.18,

6.19 e 6.20).

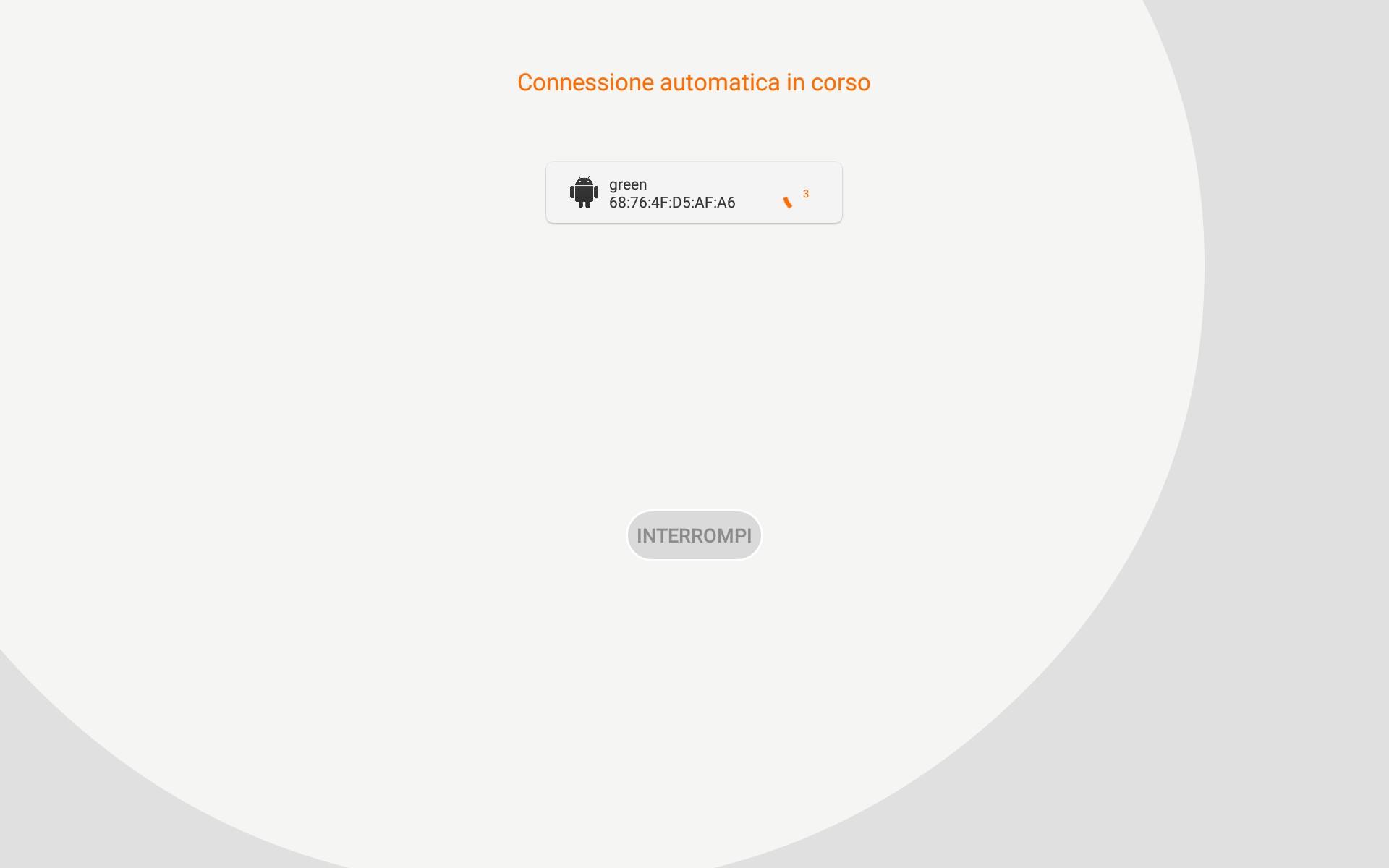


Figura 6.15: La schermata di configurazione veloce di gioco



Figura 6.16: La schermata di attesa di connessione per lo *slave*

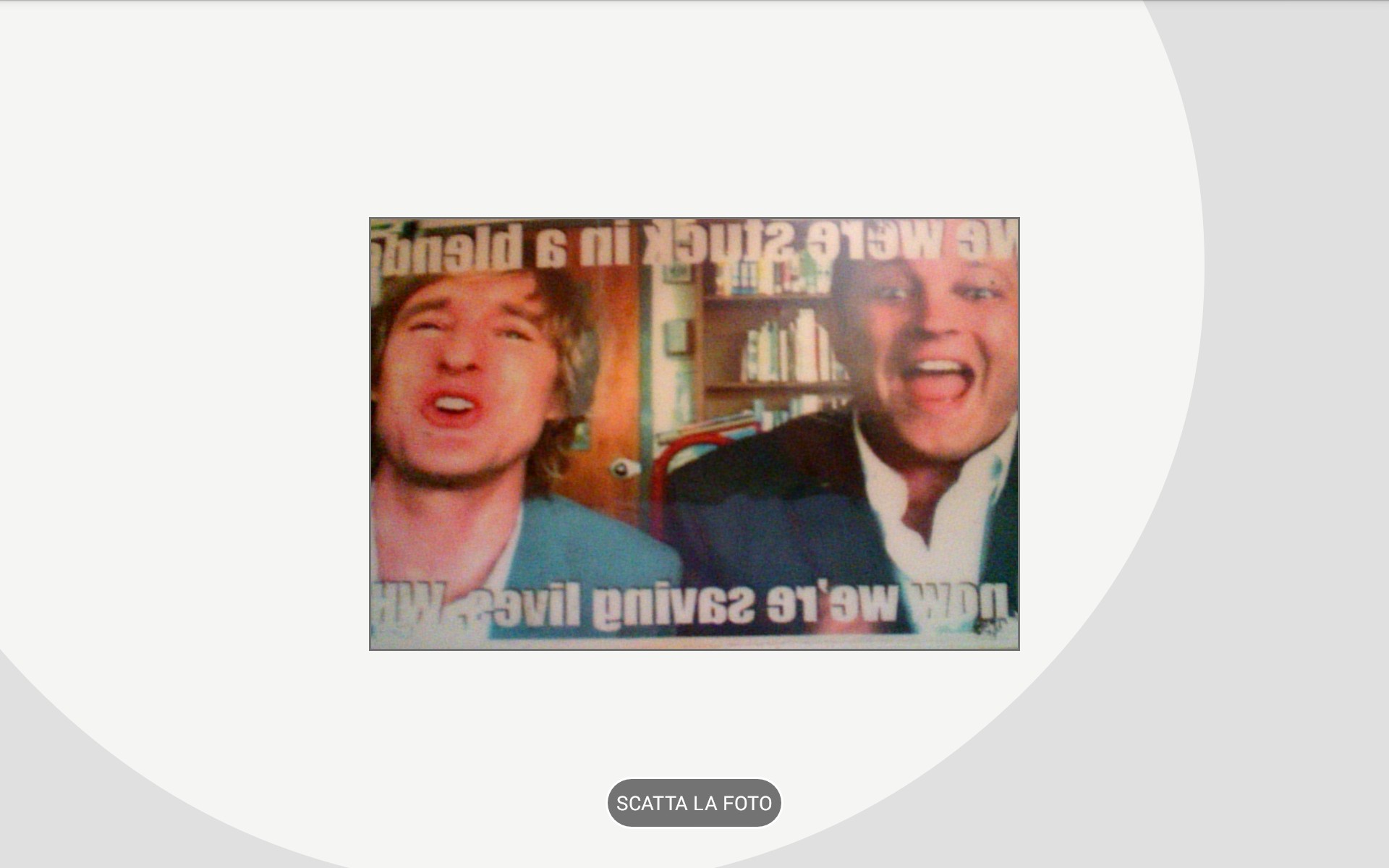


Figura 6.17: Lo scatto della foto dell’utente

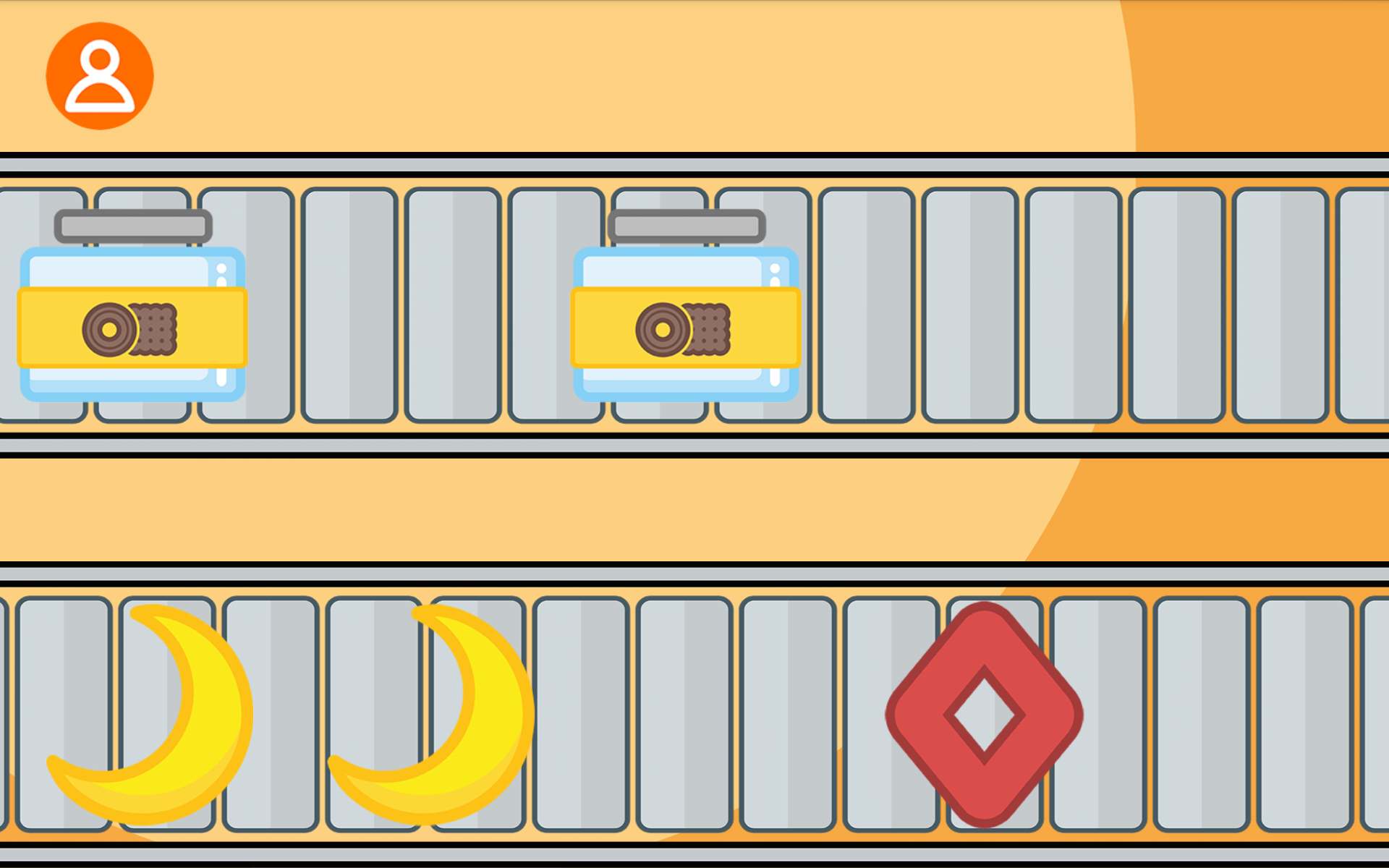


Figura 6.18: La schermata di gioco nel gioco 1

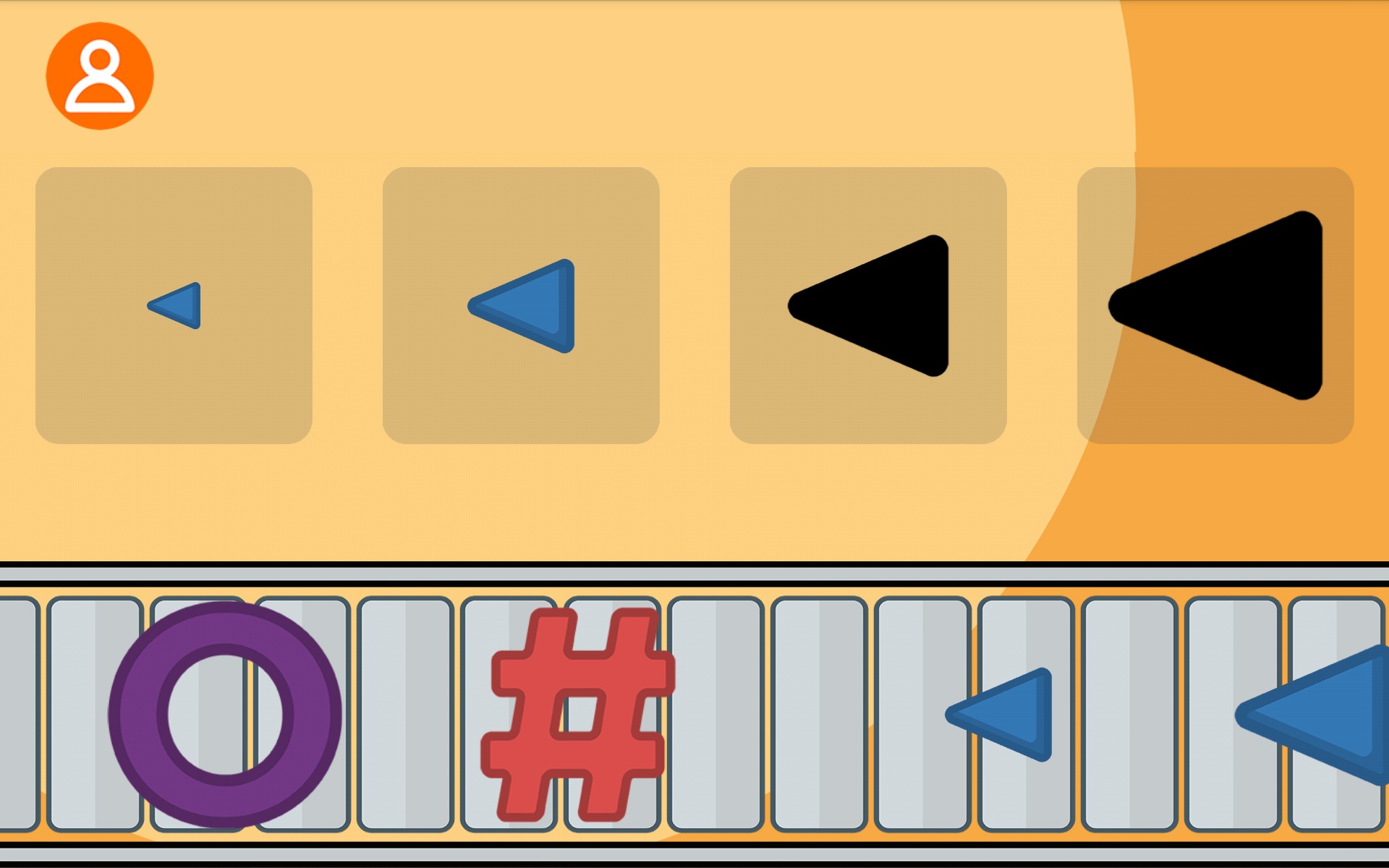


Figura 6.19: La schermata di gioco nel gioco 2



Figura 6.20: La schermata di gioco nel gioco 3