

# UNIVERSITÀ DEGLI STUDI DI SALERNO



Dipartimento di Informatica

Corso di Laurea in Informatica

## TESI DI LAUREA

### PROGETTAZIONE E SVILUPPO APP DEL GIOCO DI CARTE “BRISCOLA” SU PIATTAFORMA ANDROID

**Relatore:**  
**prof. Roberto DE PRISCO**

**Candidato/a**  
**TURA YAROSLAV**  
**Matricola: 0512/115111**

Anno Accademico 2023/2024

## Sommario

Introduzione e breve descrizione .....	4
Titolo della Tesi: <i>Sviluppo di un'Applicazione Android per il Gioco della Briscola</i> .....	4
Introduzione .....	4
Contesto e Motivazione .....	4
Regole del gioco.....	4
Carte utilizzate.....	6
Tecnologie e Strumenti Utilizzati.....	6
Progettazione dell'App .....	7
Implementazione .....	7
Modulo di Logica:.....	7
Modulo di Interfaccia Grafica:.....	7
Modulo Multiplayer:.....	7
Android.....	8
Introduzione al software .....	8
Android Studio .....	9
Installazione e preparazione .....	10
Configurazione del progetto .....	11
Caratteristiche Software.....	15
Linguaggi di Programmazione.....	15
Emulatore (AVD Manager).....	16
Scelta versione Android per Emulatore .....	18
Layout Editor.....	18
Progettazione dell'applicazione .....	20
Mockup.....	20
Implementazione .....	22
Menu.....	22
Le classi principali sono.....	29
Card.....	29
Deck .....	31
Hand .....	33
SinglePlayerActivity .....	36
Multiplayer .....	48
Librerie e strutture aggiuntive per lo scambio tra client e API.....	58
API.....	62
Test e Risultati .....	67

Conclusioni e Sviluppi Futuri .....	67
Codice sorgente.....	68
Bibliografia.....	68
Terminologia .....	68

## Introduzione e breve descrizione

**Titolo della Tesi: *Sviluppo di un'Applicazione Android per il Gioco della Briscola***

---

### Introduzione

La Briscola è uno dei giochi di carte più conosciuti e apprezzati in Italia, con una tradizione che attraversa generazioni. L'obiettivo di questa tesi è lo sviluppo di un'applicazione Android che permetta di giocare a Briscola in maniera intuitiva e coinvolgente, mantenendo il fascino del gioco tradizionale ma sfruttando le potenzialità offerte dalle moderne tecnologie mobile. Il progetto si propone di combinare un'interfaccia utente accattivante con una solida implementazione delle regole del gioco, garantendo un'esperienza piacevole sia per i principianti che per i giocatori esperti.

---

### Contesto e Motivazione

I giochi di carte hanno sempre avuto un ruolo centrale nella cultura italiana, offrendo un modo semplice ma divertente per socializzare e trascorrere il tempo. Con l'evoluzione tecnologica e la diffusione degli smartphone, i giochi tradizionali sono stati progressivamente adattati al mondo digitale. Tuttavia, esistono poche applicazioni esistenti per la Briscola.

---

### Regole del gioco

#### **Regole del Gioco della Briscola**

La **Briscola** è un gioco di carte tradizionale italiano, solitamente giocato in **due, tre o quattro giocatori** (a coppie). Si utilizza un **mazzo da 40 carte** (napoletane, siciliane, piacentine o francesi prive di 8, 9 e 10).

---

#### **1. Valore delle Carte**

Le carte hanno i seguenti valori in punti:

Carta	Punti
Asso (A)	11
Tre (3)	10
Re (K)	4
Cavallo (Q)	3

Carta	Punti
Fante (J)	2
Tutte le altre (7, 6, 5, 4, 2)	0

Il totale dei punti nel mazzo è **120**.

## 2. Preparazione del Gioco

- Si distribuiscono **tre carte a ciascun giocatore**.
- La carta successiva viene scoperta e posta sotto il mazzo: è la **Briscola** (seme dominante della partita).
- Le restanti carte formano il **mazzo di pesca**.

## 3. Svolgimento del Turno

1. Il primo giocatore gioca una carta a sua scelta.
2. Gli altri giocatori, in senso orario, giocano una carta senza obbligo di seguire il seme.
3. Vince la mano:
  - La carta di **Briscola** più alta.
  - Se nessuna Briscola è giocata, la carta più alta del **seme della prima carta giocata**.
4. Il vincitore della mano prende le carte e pesca per primo.
5. Il gioco prosegue fino all'esaurimento delle carte del mazzo.

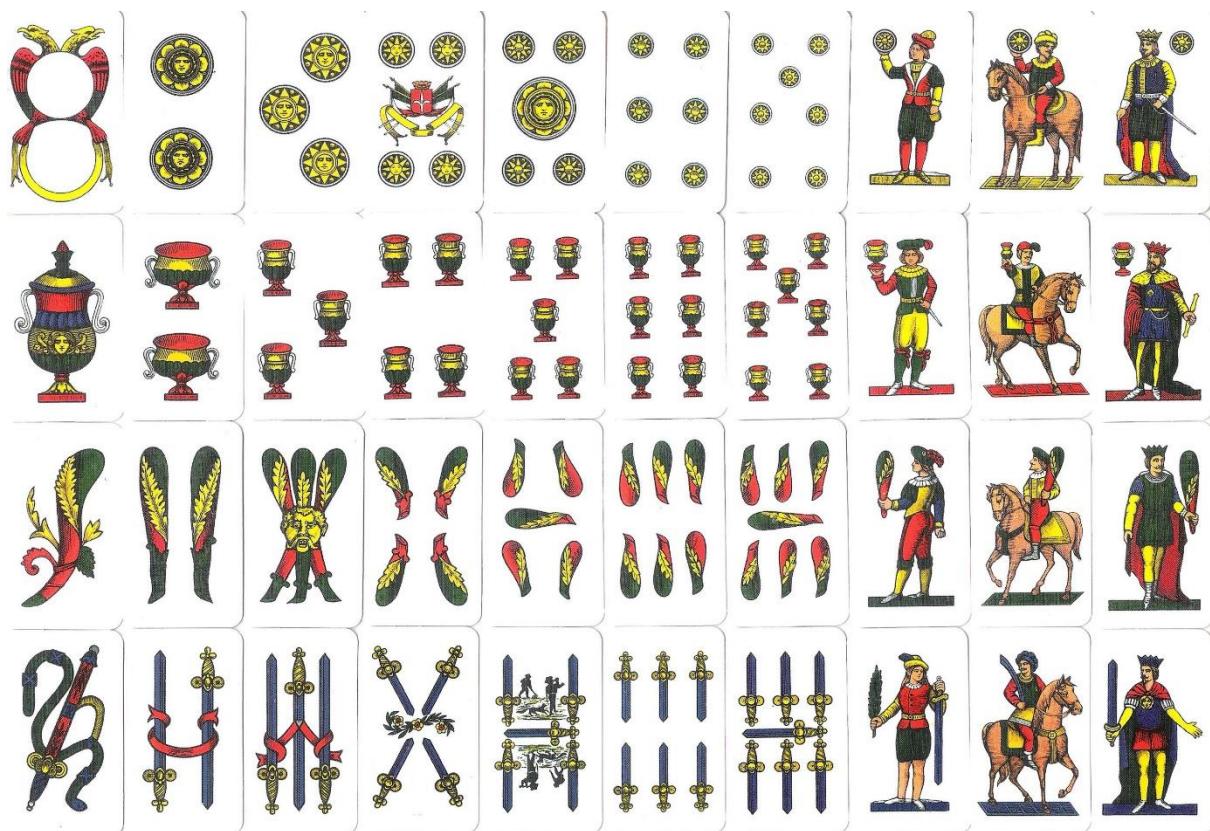
## 4. Fine della Partita e Punteggio

- Al termine delle mani, ogni giocatore (o coppia) somma i punti delle carte vinte.
- Vince chi ha il punteggio più alto (**oltre 60 punti su 120**).
- In caso di pareggio a **60-60**, si ripete la partita.

## 5. Varianti

- **Briscola in 5 (Briscola Chiamata)**: Si gioca in 5, con un giocatore segreto che si allea con chi ha una carta specifica.
- **Briscola Scoperta**: Tutti i giocatori giocano con le carte scoperte.

## Carte utilizzate



## Tecnologie e Strumenti Utilizzati

Lo sviluppo dell'applicazione è stato realizzato utilizzando le seguenti tecnologie e strumenti:

- **Android Studio:** Ambiente di sviluppo integrato (IDE) ufficiale per lo sviluppo di app Android.
- **IntelliJ IDEA:** Ambiente di sviluppo per la nostra API
- **Java:** Linguaggio di programmazione scelto per la sua robustezza e ampia diffusione nel contesto Android.
- **XML:** Per il design delle interfacce utente.
- **Material Design:** Linee guida di Google per creare interfacce utente intuitive e visivamente piacevoli.
- **Spring Boot:** un framework Java open source utilizzato per sviluppo API
- **Volley:** libreria per android per le chiamate JSON

## Progettazione dell'App

L'applicazione è stata progettata seguendo il paradigma architetturale **MVC** (Model-View-Control), che separa chiaramente i componenti logici, visivi e di gestione dei dati. La progettazione dell'app si è focalizzata sui seguenti aspetti:

1. **Logica di Gioco:** Implementazione delle regole tradizionali della Briscola, inclusa la gestione del mazzo, dei punteggi e dei turni.
  2. **Interfaccia Utente (UI):** Creazione di una grafica accattivante, ispirata ai mazzi di carte regionali italiani.
  3. **Esperienza Utente (UX):** Progettazione di un'interazione semplice e fluida
- 

## Implementazione

L'app è composta da diversi moduli che lavorano insieme per offrire un'esperienza di gioco completa:

### Modulo di Logica:

- Generazione e gestione del mazzo di carte.
- Controllo delle regole del gioco, inclusi i criteri di assegnazione dei punti e la determinazione del vincitore.

### Modulo di Interfaccia Grafica:

- Layout dinamici per l'adattamento a diverse dimensioni di schermo.
- Animazioni fluide per rappresentare le mosse dei giocatori

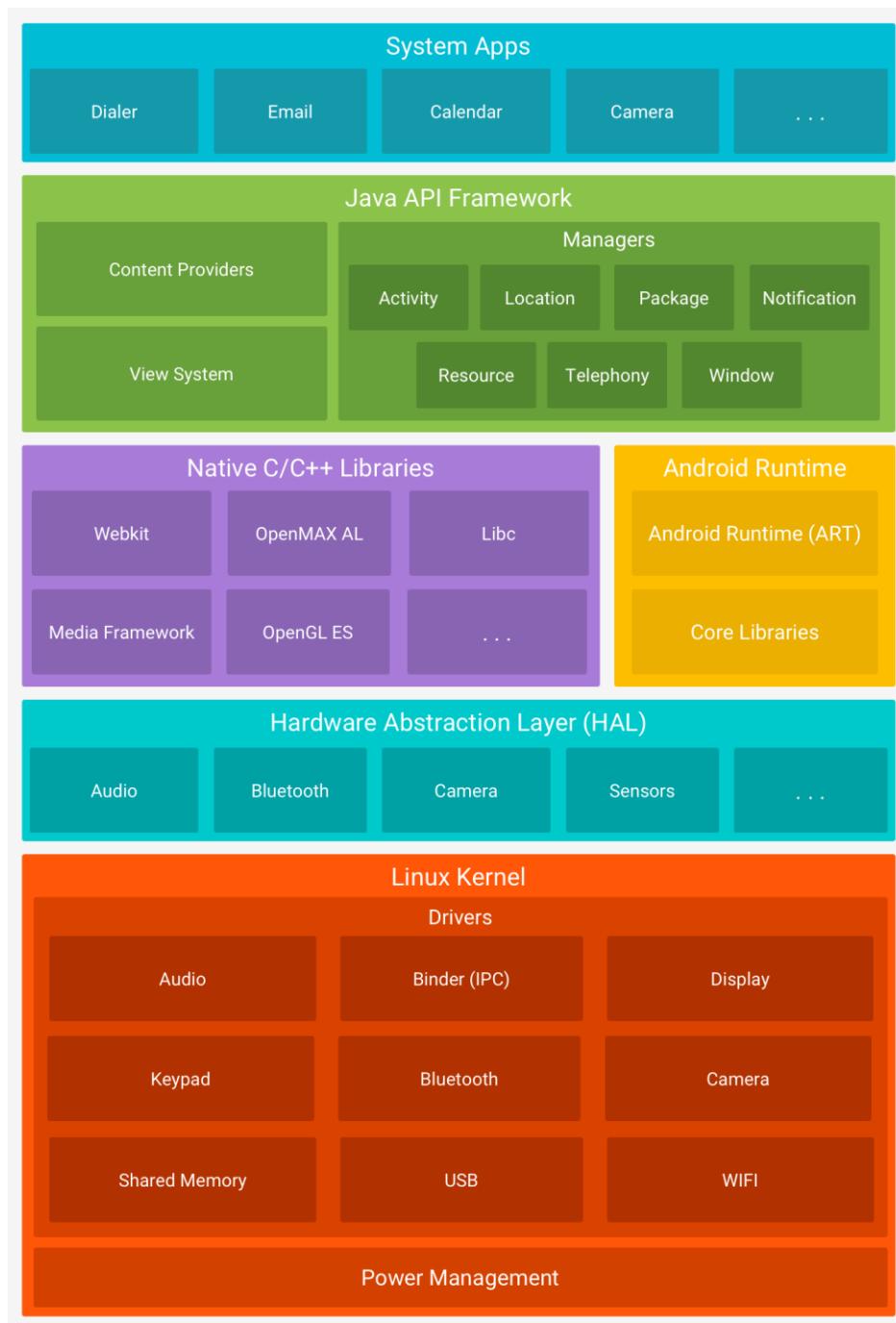
### Modulo Multiplayer:

- Funzioni per interagire con altri giocatori mediante API
-

# Android

## **Introduzione al software**

Android è un sistema operativo per dispositivi mobili basato su una versione modificata del kernel di Linux e di altri open source software. Principalmente viene usato su dispositivi abilitati con touchscreen quali smartphone e tablet. Il suo codice sorgente è stato utilizzato su vari dispositivi, quali camere digitali, PC e game console, ognuna con, ovviamente, un'interfaccia diversa.



## Android Studio

Android Studio è l'IDE (Integrated Development Environment) ufficiale per il sistema operativo Android basato sul software IntelliJ IDEA di JetBrains e progettato specificamente per lo sviluppo Android esso è disponibile per il download su Windows, macOS e Linux. Per lo sviluppo di applicazioni nel sistema operativo Android, Android Studio utilizza un sistema di compilazione basato su Gradle, un emulatore, un template di codice e l'integrazione di GitHub. Android Studio utilizza una funzione Instant Push

per inviare modifiche di codice e risorse a un'applicazione in esecuzione. Un editor di codice assiste lo sviluppatore nella scrittura del codice e nell'offrire completamento, rifrazione e analisi del codice.

## Installazione e preparazione

Inizialmente bisogna scaricare il software direttamente dal sito Android. Una volta installato sul proprio dispositivo, si procede alla preparazione dell'IDE.

Come possiamo vedere nella figura 2.1, abbiamo, da una parte, la lista dei nostri progetti implementati nell'applicazione e nella parte principale, una lista di opzioni. Tra le varie opzioni si può scegliere quella di aprire un progetto esistente o di importarlo dal nostro dispositivo, nonché, ovviamente, quella di iniziare un nuovo progetto. Quindi creiamo un nuovo progetto e accediamo alla schermata in Figura 2.2:

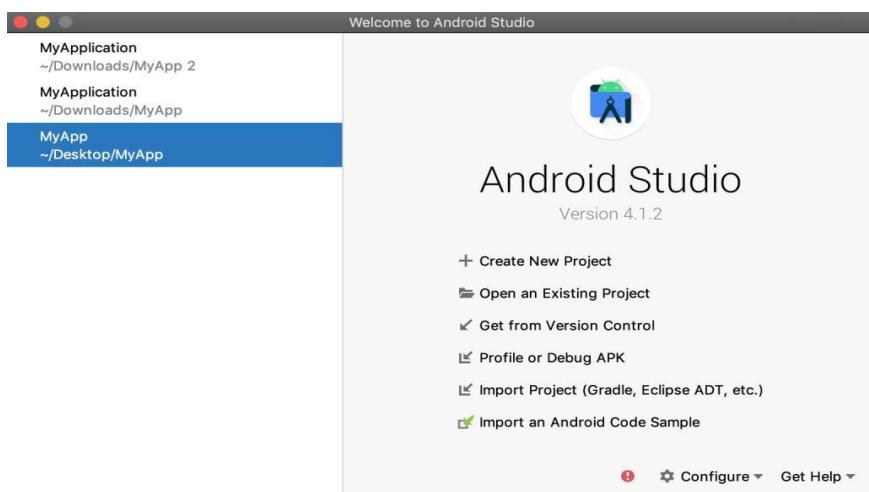


Figura 2.1: Schermata iniziale di Android Studio

Nella prossima immagine Figura 2.2 ci viene presentata una serie di scelte in base al tipo di applicazione che vogliamo implementare. Inizialmente, come vediamo, possiamo implementare applicazioni per diversi dispositivi: smartphone, smartwatch, TV Android etc. Una volta effettuata tale scelta, che, nel caso della nostra applicazione riguarderà smartphone e tablet, ci viene chiesto di effettuare un'altra scelta sul tipo di interfaccia che debba avere la nostra activity. Android gestisce questo tipo di interfaccia in maniera diversa, per questo conviene avere un'interfaccia vuota in modo da avere il controllo sul nostro codice.

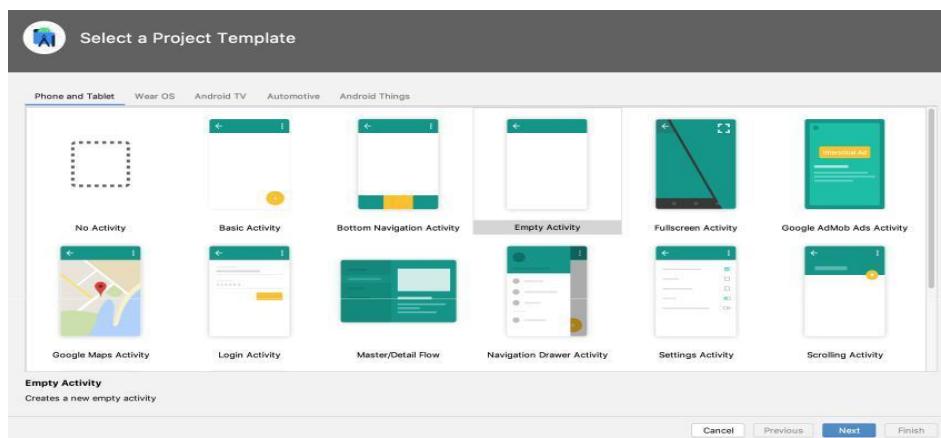


Figura 2.2: Template del Progetto

## Configurazione del progetto

Ora Android Studio ci chiede il nome del nostro progetto e il tipo di linguaggio di programmazione che si vuole utilizzare per la codificazione della nostra applicazione. Noi abbiamo optato per il linguaggio Java che approfondiremo seguito. Poi è necessario scegliere l'API minima che supporterà la nostra applicazione.

Ovviamente più aumenteremo il livello dell'API, minori saranno i dispositivi su cui la nostra applicazione sarà supportata come si vede in figura 2.3.

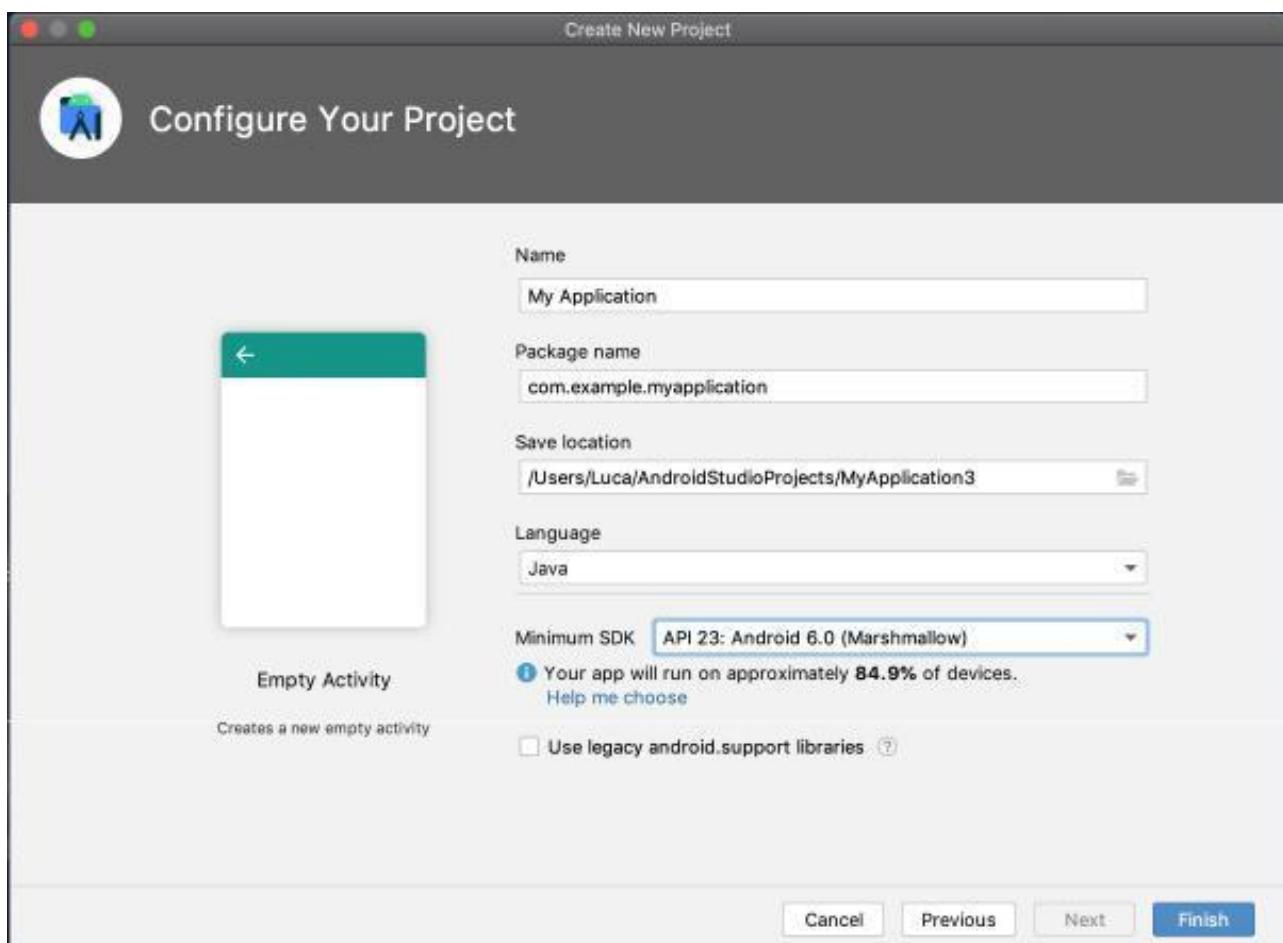


Figura 2.3: Configurazione del progetto

Nella Figura 2.4 vediamo come, una volta terminata la configurazione, arriviamo alla schermata della struttura, dove visualizziamo il nostro programma. Abbiamo, come primo elemento, il modulo "app" suddiviso in 3 cartelle: Manifest, java, res. Per quanto riguarda la cartella Manifest, al suo interno troveremo, appunto, il file `AndroidManifest.xml` definisce le proprietà della nostra applicazione, tra le proprietà definite in tale file abbiamo: Il package, ovvero l'ID univoco dell'applicazione; un tag "application" con all'interno, le varie proprietà della nostra applicazione, quali la forma dell'icona, il nome, il tema, e l'elenco delle nostre activity con alcune proprietà, come, ad esempio, la definizione della nostra main activity o di quale activity è "parente" di un'altra. Poi, nella cartella java, abbiamo i file della `MainActivity`, nella quale, durante l'implementazione, andremo a inserire le varie activity che andranno a comporre la nostra

applicazione. L'ultima cartella è quella delle risorse o res, dove abbiamo varie sottocartelle che conterranno file molto importanti per l'implementazione. In questa cartella andremo a immettere i layout delle varie activity, le immagini che vogliamo inserire, come, ad esempio, il logo dell'applicazione, o values, dove possiamo inserire le stringhe, i colori o gli stili.

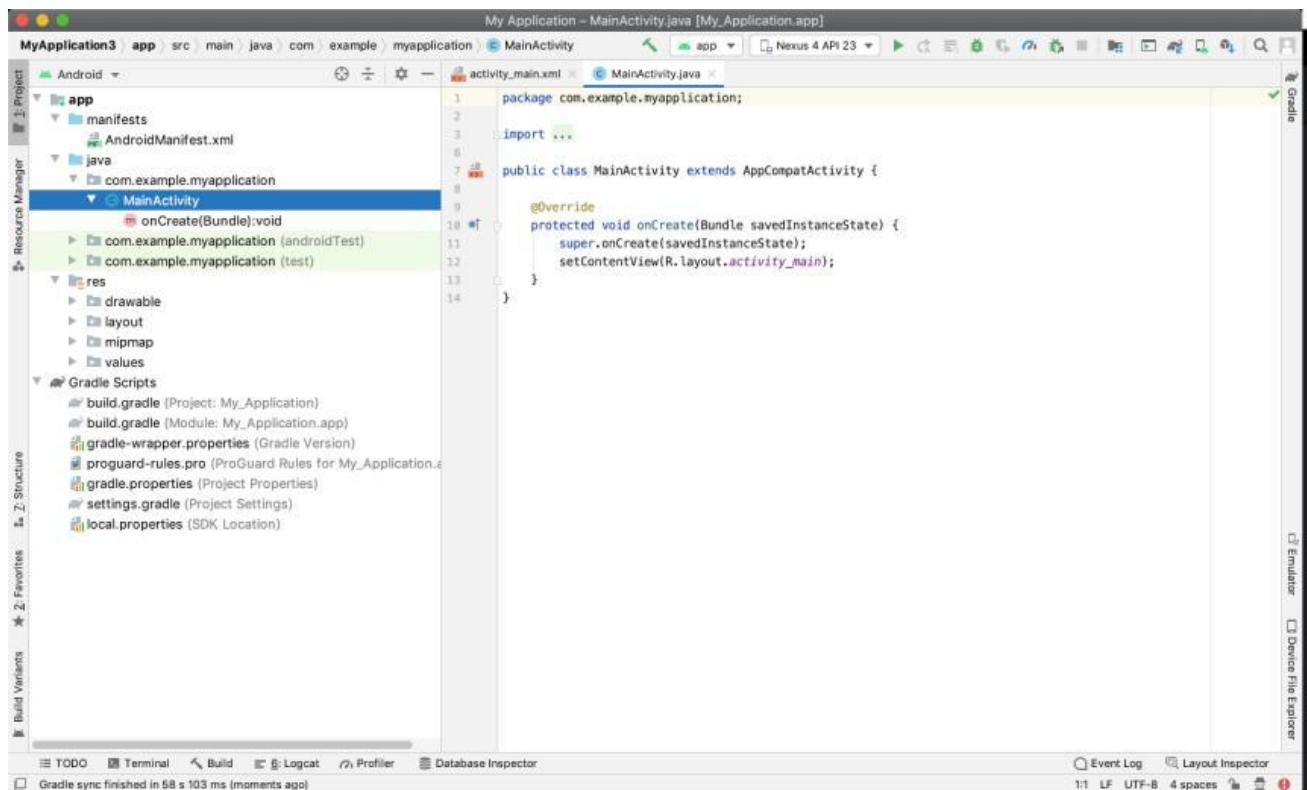


Figura 2.4: Struttura del progetto

## AndroidManifest.XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools">
4      <uses-permission android:name = "android.permission.ACCESS_WIFI_STATE" />
5      <uses-permission android:name = "android.permission.ACCESS_NETWORK_STATE" />
6      <uses-permission android:name = "android.permission.INTERNET"/>
7
8      <application
9          android:allowBackup="true"
10         android:dataExtractionRules="@xml/data_extraction_rules"
11         android:fullBackupContent="@xml/backup_rules"
12         android:icon="@mipmap/ic_launcher"
13         android:label="Briscola"
14         android:roundIcon="@mipmap/ic_launcher_round"
15         android:supportsRtl="true"
16         android:theme="@style/Theme.Test"
17         android:usesCleartextTraffic="true"
18         android:networkSecurityConfig="@xml/network_security_config"
19
20
21         tools:targetApi="31">
22
23             <activity
24                 android:name=".MenuOnline2"
25                 android:exported="false" />
26             <activity
27                 android:name=".MenuOnlinenv2"
28                 android:exported="false" />
29             <activity
30                 android:name=".SinglePlayerActivity2"
```

App richiede 3 permessi per poter funzionare con il multiplayer

Inoltre in App è stata disattivata la rotazione del dispositivo in quanto non è previsto che interfaccia abbia rapporto dei lati dello schermo diverso da quello in modalità portrait

```
29      <activity
30          android:name=".SinglePlayerActivity2"
31          android:exported="false" />
32      <activity
33          android:name=".SinglePlayerActivity"
34          android:exported="false" />
35      <activity
36          android:name=".Multiplayer"
37          android:exported="false" />
38      <activity
39          android:name=".Multiplayerv2"
40          android:exported="false" />
41      <activity
42          android:name=".MenuPrincipale"
43          android:exported="true"
44          android:screenOrientation="portrait"
45          android:theme="@style/Theme.AppCompat.Light.NoActionBar.FullScreen">
46          <intent-filter>
47              <action android:name="android.intent.action.MAIN" />
48
49              <category android:name="android.intent.category.LAUNCHER" />
50          </intent-filter>
51      </activity>
52  </application>
53
54</manifest>
```

## Caratteristiche Software

Android Studio è un IDE che ha acquisito sempre più valore negli ultimi anni anche grazie alle varie feature che esso prevede.

## Linguaggi di Programmazione

Come abbiamo visto in precedenza, Android Studio ci dà la possibilità di scegliere tra vari linguaggi di programmazione per la codificazione della nostra applicazione, tra i quali abbiamo Kotlin e Java. Nel nostro caso si è utilizzato il linguaggio Java. Java è un linguaggio di programmazione avanzato orientato agli oggetti, progettato per avere indipendenza

dall'hardware su cui si andrà a eseguire il nostro codice, infatti uno, dei principi fondamentali di questo linguaggio è "write once, run anywhere", ovvero "scrivi una volta sola e esegui ovunque". Una volta scritto il nostro codice, esso non dovrà essere modificato se cambia il dispositivo su cui si vuole eseguirlo.

## Emulatore (AVD Manager)

L'emulatore all'interno di Android Studio è una delle feature più importanti dell'IDE, infatti esso ci permette di testare le nostre applicazione su vari dispositivi con API di livelli diversi senza dover prendere il corrispettivo dispositivo fisico. L'Emulatore è in grado di simulare quasi tutte le caratteristiche di un dispositivo, infatti esso è in grado di simulare chiamate in entrata e uscita, come pure la rotazione del dispositivo,in modo da testare se la nostra applicazione si adatta alle varie grandezze di schermo e molto altro. Andiamo ora a vedere come impostare un dispositivo che ci permetterà di testare la nostra applicazione. Una volta aperto l'AVD manager, ci troveremo di fronte alla schermata in Figura 2.5

Qui possiamo vedere una lista dei dispositivi già creati e l'opzione di crearne uno nuovo. Vediamo ora come crearne uno da zero.

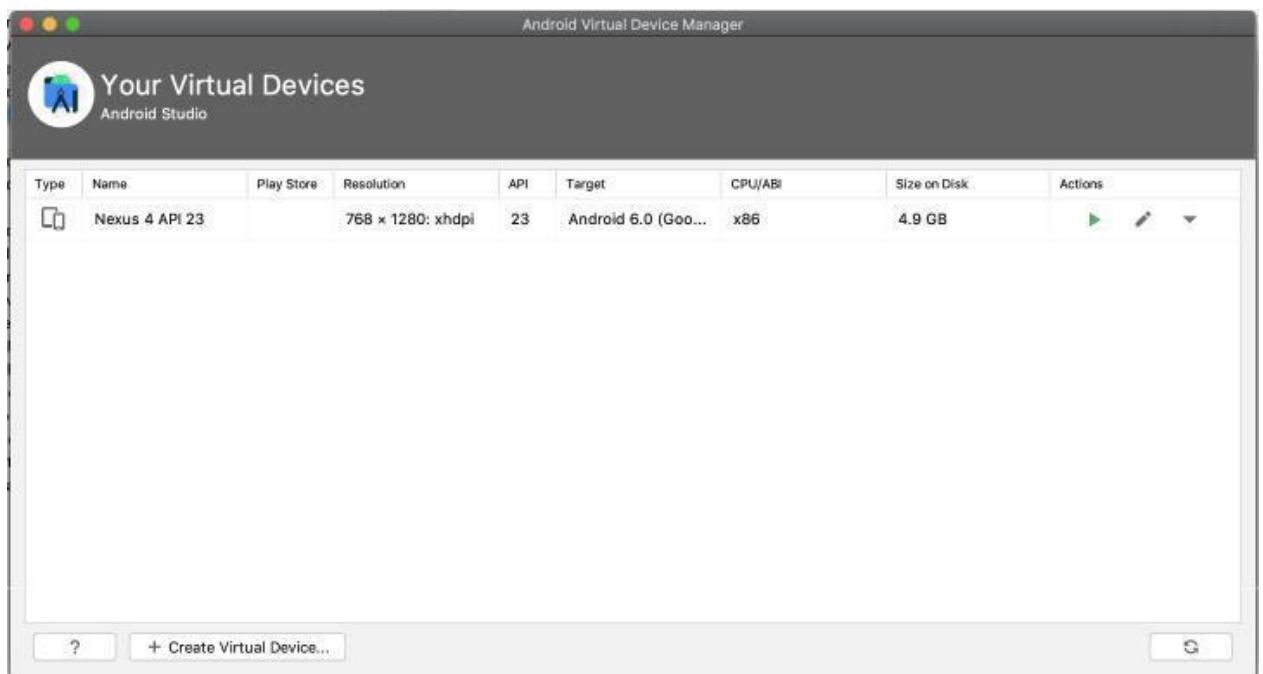


Figura 2.5: Creazione di un nuovo dispositivo

Nella Figura 2.6 vediamo come scegliere il tipo di dispositivo che vogliamo simulare; a sinistra abbiamo un elenco di tipologie di dispositivi, dagli smartphone alle tv etc, (come abbiamo già detto noi ci occuperemo di smartphone e tablet). Quindi ora abbiamo una serie di dispositivi ciascuno con la propria grandezza e la propria risoluzione dello schermo, l'icona che ci permette di individuare i dispositivi che supportano le librerie di Google Play, in particolare quella del Play Services. Una volta effettuata questa scelta, ci troveremo di fronte alla schermata in Figura 2.7.

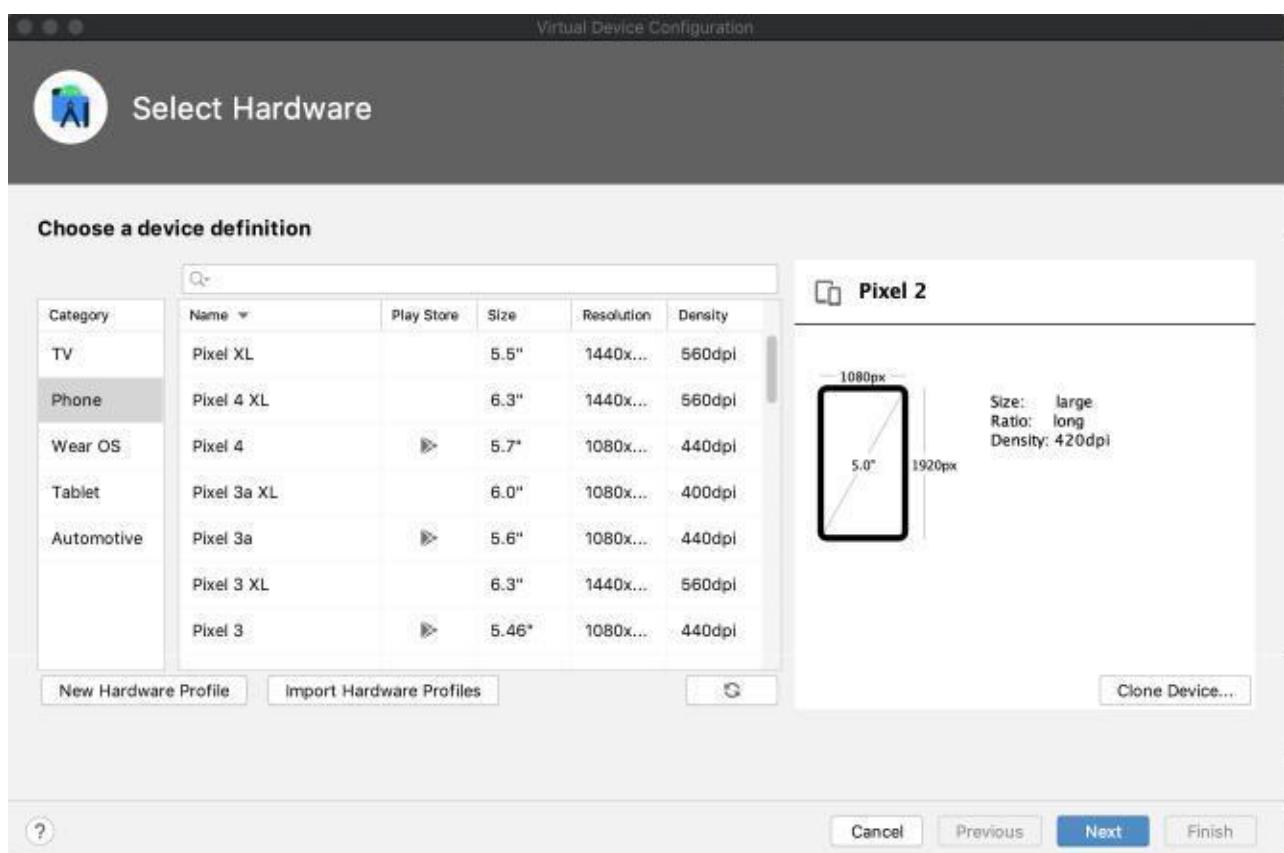


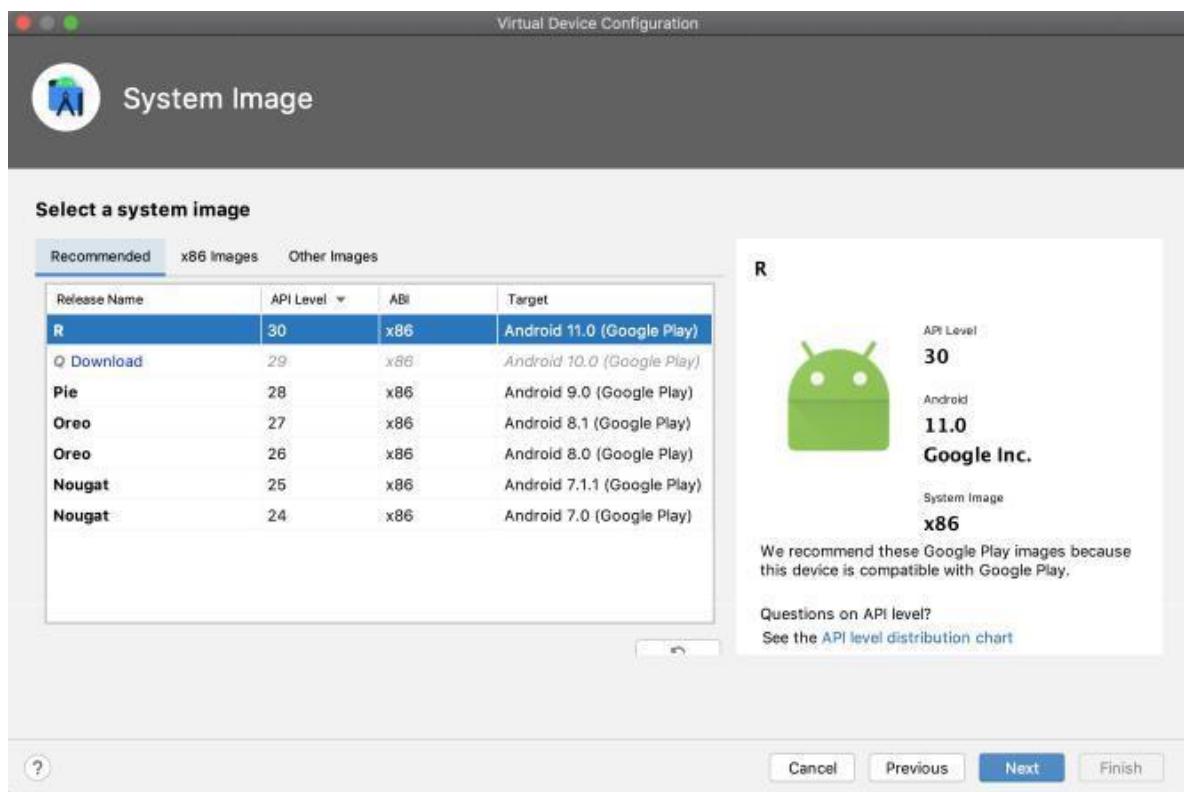
Figura 2.6: Scelta del dispositivo da simulare

## Scelta versione Android per Emulatore

Figura 2.7: Scelta dell'API minima che dovrà essere supportata dal dispositivo

Qui potremo decidere l'API minima che dovrà supportare il nostro dispositivo. In base alla nostra scelta ci verrà permesso di scaricare le opportune librerie. Una volta fatto ciò e dato un nome al nostro emulatore, il programma è pronto per essere utilizzato.

In questo progetto è stato utilizzato API 31 che corrisponde a Android 12. Non c'è una ragione specifica per la scelta , è stato consigliato da Android Studio che ha fornito anche la statistica di dispositivi in cui risultava un buon percentuale di dispositivi con API 31 o superiori.



## Layout Editor

Il layout editor è uno degli strumenti fondamentali per la costruzione dei layout delle varie schermate della nostra applicazione. Esso, oltre alla

codifica manuale di file XML di tali layout, ci permette di trascinare elementi per la creazione dell'interfaccia in un editor per la progettazione visiva del layout. Andiamo, ora, a vedere le varie features di questo editor nella Figura 2.8.

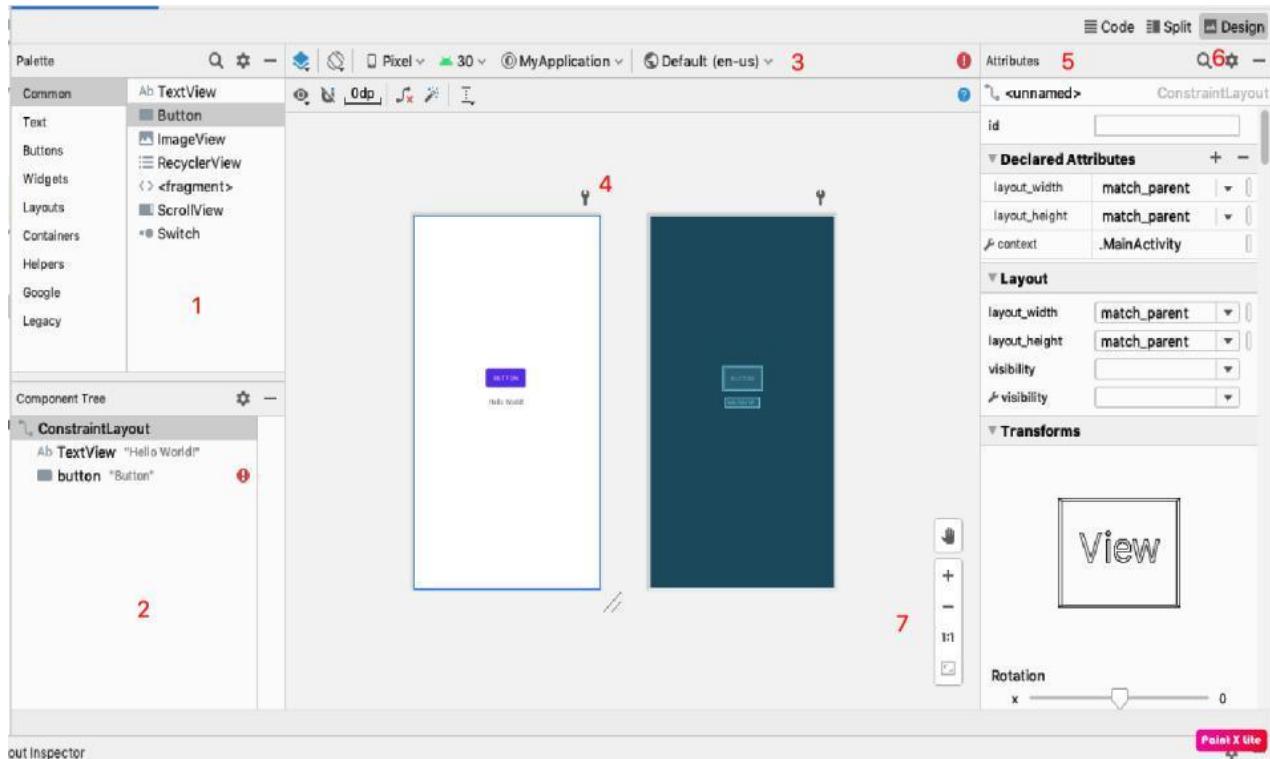


Figura 2.8: Il Layout Editor

# Progettazione dell'applicazione

## Mockup

Mockup è uno degli strumenti utilizzati per la progettazione di un'app , in particolare si usa per le riproduzioni grafiche di oggetti.

In questo caso prima di implementare qualsiasi cosa va progettata interfaccia e proprio grazie a mockup è possibile già vedere che aspetto potrebbe avere il prodotto finale e fare determinate scelte , semplifica molto lo sviluppo Una volta progettata interfaccia si passa alla implementazione

### 4 Mockup principali



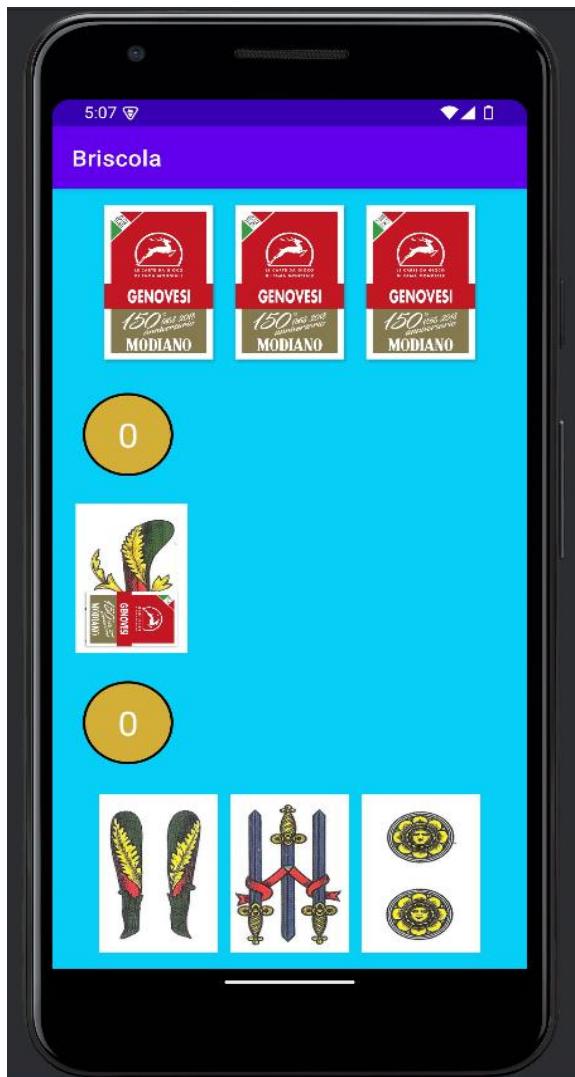
**Menu principale**



**Sub Menu singolo**



**Sub Menu Multiplayer**



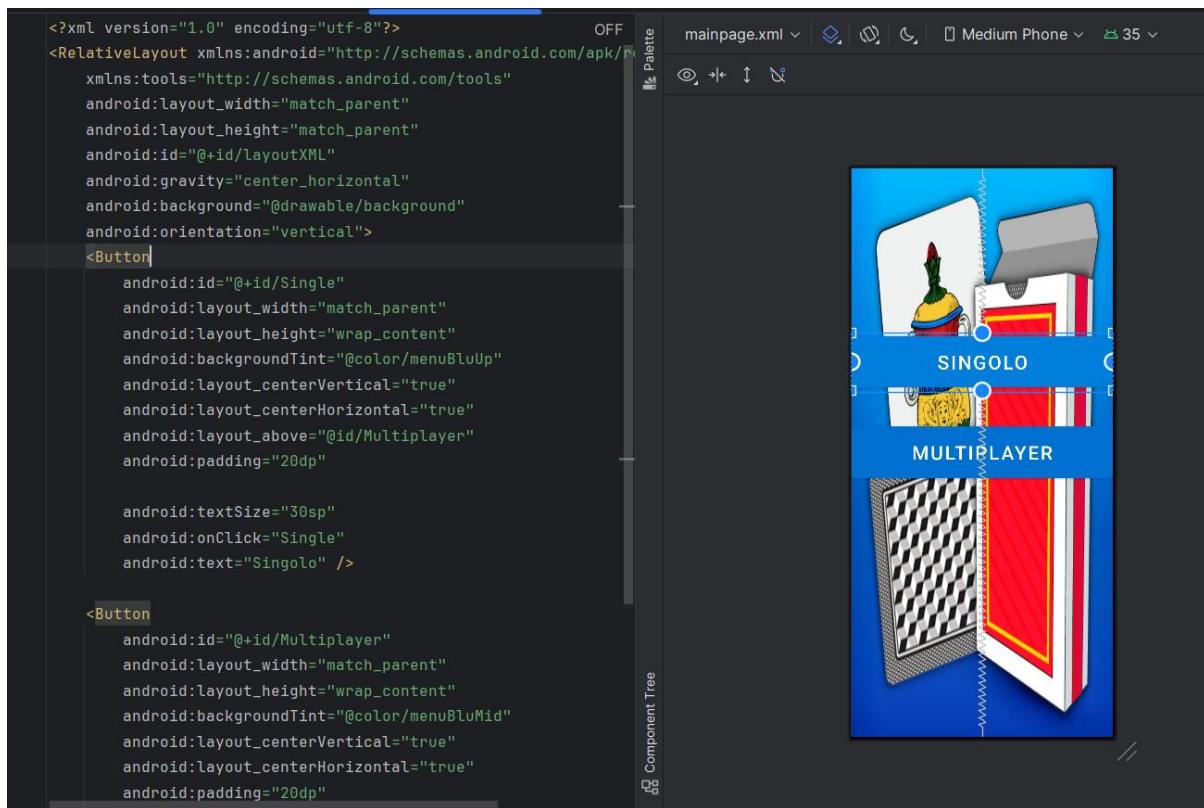
**Interfaccia Partita**

# Implementazione

## Menu

### Menu principale

Menu sono activity semplici che sono composte da due oggetti Button e un TextView per visualizzare eventuali messaggi in caso di necessità . Ogni Button possiede proprietà onClick() che è legata ad una funzione che viene chiamata quando si preme sul botton. è stato utilizzato Relativelayout per la sua semplicità ed relativa libertà di posizionamento degli elementi. In questo caso il 2 Button è stato posizionato al centro rispetto l'asse Y ed il 1 Button è stato posizionato sopra.



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/layoutXML"
    android:gravity="center_horizontal"
    android:background="@drawable/background"
    android:orientation="vertical">
    <Button
        android:id="@+id/Single"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/menuBluUp"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:layout_above="@+id/Multiplayer"
        android:padding="20dp"
        android:textSize="30sp"
        android:onClick="Single"
        android:text="Singolo" />
    <Button
        android:id="@+id/Multiplayer"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/menuBluMid"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:padding="20dp" />

```

La classe MenuPrincipale che definisce le funzioni del layout definito sopra.

OnCreate con il DisplayMetric e WindowManager possiamo ottenere le misure dello schermo , questo ci permette di modificare elementi di layout in base DPI o dimensione dello schermo.

```
11 ></> public class MenuPrincipale extends AppCompatActivity {
12     1 usage
13     Button Single, Multiplayer;
14     protected void onCreate(Bundle savedInstanceState) {
15
16         super.onCreate(savedInstanceState);
17         setContentView(R.layout.mainpage);
18
19         Single = findViewById(R.id.Single);
20         Multiplayer = findViewById(R.id.Multiplayer);
21
22         DisplayMetrics displayMetrics = new DisplayMetrics();
23         getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);
24         int height = displayMetrics.heightPixels;
25         int width = displayMetrics.widthPixels;
26
27
28         DisplayMetrics metrics = getResources().getDisplayMetrics();
29         float density = metrics.density; // Ottieni il fattore di scala della densità
30
31     }
32 }
```

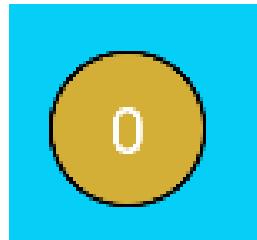
Poi abbiamo due funzioni che sono legati ai Button, la prima crea Intent dove un Intent in Android è un oggetto utilizzato per la comunicazione tra componenti di un'applicazione o tra diverse applicazioni. Spesso è usato per immagazzinare i dati che poi vengono recuperati quando passiamo alla prossima activity. In caso non serve immagazzinare niente di extra , quindi creiamo uno con due parametri la classe attuale e la classe a cui vogliamo passare. Dopo con finish() finiamo l'attività attuale e subito dopo viene lanciata la nuova activity con startActivity(i)

```
33     no usages
34     public void Single(View v){
35         Intent i = new Intent( packageContext: MenuPrincipale.this, SinglePlayerActivity2.class);
36         finish(); //Kill the activity from which you will go to next activity
37         startActivity(i);
38     }
39
40     1 usage
41     public void Multiplayer(View v){
42         Intent i = new Intent( packageContext: MenuPrincipale.this, MenuOnlinev2.class);
43         finish(); //Kill the activity from which you will go to next activity
44         startActivity(i);
45     }
```

## Menu partita Singolo



```
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:id="@+id/layoutR"
8     android:background="@color/bluCustom">
9
10    <ImageView
11        android:id="@+id/im1"
12        android:layout_toLeftOf="@+id/im2"
13        android:layout_width="220dp"
14        android:layout_height="295dp"
15        android:layout_margin="5dp"
16        app:srcCompat="@drawable/r2" />
17
18    <ImageView
19        android:id="@+id/im2"
20        android:layout_centerHorizontal="true"
21        android:layout_width="220dp"
22        android:layout_height="295dp"
23        android:layout_margin="5dp"
24        app:srcCompat="@drawable/r2" />
25
26    <ImageView
27        android:id="@+id/im3"
28        android:layout_toRightOf="@+id/im2"
29        android:layout_width="220dp"
30        android:layout_height="295dp"
31        android:layout_margin="5dp"
32        app:srcCompat="@drawable/r2" />
```



Definisce elemento il cerchio giallo con il testo all'interno

```

896     private void newGame(boolean result,int pc ,int player){
897         //      RelativeLayout layout = (RelativeLayout) findViewById(R.id.layoutR);
898         //      if(result) layout.setBackgroundColor(getColor(R.color.greenVictory));
899         //      else layout.setBackgroundColor(getColor(R.color.redLose));
900         t2.setVisibility(View.INVISIBLE);
901         tx1.setVisibility(View.INVISIBLE);
902         tx2.setVisibility(View.INVISIBLE);
903         tx3.setVisibility(View.INVISIBLE);
904         t0.setVisibility(View.INVISIBLE);
905         t1.setVisibility(View.INVISIBLE);
906         testoSopra.setText("");
907         testoGiu.setText("");
908         FrameTop.setVisibility(View.INVISIBLE);
909         FrameDown.setVisibility(View.INVISIBLE);
910         ng.setVisibility(View.VISIBLE);
911         menuB.setVisibility(View.VISIBLE);
912         ResultText.setVisibility(View.VISIBLE);
913         ResultText.setText("Punteggio:\n pc "+pc+"\n giocatore "+player);
914         ng.setOnClickListener(new View.OnClickListener() {
915             @Override
916             ↑
917             public void onClick(View view) {
918                 Intent intent = getIntent();
919                 finish();
920                 startActivityForResult(intent);
921             });
922         menuB.setOnClickListener(new View.OnClickListener() {
923             ↑
924             @Override
925             public void onClick(View view) {
926                 Intent i = new Intent( packageContext: SinglePlayerActivity2.this, MenuPrincipale.class);
927                 finish(); //Kill the activity from which you will go to next activity
928                 startActivityForResult(i);
929             });

```

Il codice che gestisce i due tasti “Nuova partita” e “Torna al menu”.

Con View.Invisible togliamo la visibilità agli elementi senza toglierli in modo tale che altri componenti restano ai loro posti senza muoversi

## Menu Online

```
11
12     <Button
13         android:id="@+id/trovaPartita"
14         android:layout_width="match_parent"
15         android:layout_height="wrap_content"
16         android:backgroundTint="@color/menuBluUp"
17         android:textColor="@color/white"
18         android:layout_centerVertical="true"
19         android:layout_above="@+id/TornaMenu"
20         android:padding="20dp"
21         android:textSize="30sp"
22         android:onClick="FindGame"
23         android:text="Trova partita" />
24
25     <Button
26         android:id="@+id/TornaMenu"
27         android:layout_width="match_parent"
28         android:layout_height="wrap_content"
29         android:backgroundTint="@color/menuBluMid"
30         android:textColor="@color/white"
31         android:layout_centerVertical="true"
32         android:padding="20dp"
33         android:textSize="30sp"
34         android:layout_marginVertical="50dp"
35         android:onClick="BackToMenu"
36         android:text="Torna al Menu" />
37
38     <TextView
39         android:id="@+id/Avviso"
40         android:layout_width="match_parent"
```



Composto da 3 elementi 2 button e 1 TextView

Ha il ruolo importante nella gestione online. Quando si preme il tasto trova partita viene recuperato il nostro ID del dispositivo – che viene assegnato quando un'app viene installata , quindi reinstallando app cambierà ID

```

75      public void FindGame(View view) throws IOException {
76
77          //FindGame.setEnabled(false);
78
79
80          @SuppressLint("HardwareIds") String mId = Settings.Secure.getString(getApplicationContext(), Settings.Secure.ANDROID_ID);
81
82          RequestQueue requestQueue = Volley.newRequestQueue( context: this);
83          String url1 = url + "ready";
84          Uri.Builder builder = Uri.parse(url1).buildUpon();
85          builder.appendQueryParameter("idp", mId);
86          Log.d( tag: "id1",mId);
87          //builder.appendQueryParameter("param2", "value2");
88          String urlWithParams = builder.build().toString();
89
90          messageOnDisplay.setVisibility(View.VISIBLE);
91
92          StringRequest stringRequest = new StringRequest(
93              Request.Method.GET, urlWithParams,
94              new Response.Listener<String>() {
95                  no usages
96                  @Override
97                  public void onResponse(String response) {
98                      // Gestisci la risposta come stringa
99                      if (response.contentEquals( cs: "new")){
100                          messageOnDisplay.setText("Connesso... aspetta inizio partita!");
101                          startNewGame();
102                      }
103                      if (!response.contentEquals( cs: "new")){
104                          Intent i = new Intent( packageContext: MenuOnline2.this, Multiplayer.class);
105                          i.putExtra( name: "idPlayer", mId);
106                          i.putExtra( name: "idPartita",response);
107
108                          finish(); //Kill the activity from which you will go to next activity
109                          startActivity(i);
110
111                      else
112                          messageOnDisplay.setText("Errore... probabilmente non hai connessione a Internet!");
113                  }
114                  new Response.ErrorListener() {
115                      no usages
116                      @Override
117                      public void onErrorResponse(VolleyError error) {
118                          // Gestisci eventuali errori
119                          Log.e( tag: "VolleyError", msg: "Errore: " + error.getMessage());
120                      }
121                  });
122
requestQueue.add(stringRequest);

```

**La funzione FindGame ha la seguente funzione :**

**Viene recuperato ID del utente , usando la libreria Volley viene assegnata la nostra futura richiesta alla coda . La nostra richiesta è composta dal indirizzo base url = <http://localhost:8080/post/> + ready che il endpoint per la chiamata API su nostro server. In più sono assegnati anche parametri addizionali con builder.appendQueryParameter("idp", mId);**

**Una volta creati parametri sopra citati possiamo procedere con la richiesta:**

**con StringRequest viene chiamato il metodo che in questo caso usa metodo GET trasmettendo dei parametri in risposta ottiene una stringa dopo che la richiesta è stata elaborata dal server che ha risposto con una stringa. Ho gestito in seguente modo se il server restituisce “new” allora siamo in numero dispari in quanto sul server c’è la coda che funziona in modo che ogni**

**richiesta viene contata e se il contatore è modulo 2 ossia pari allora c'è numero sufficiente di giocatori ossia almeno 2 per poter iniziare la partita vengono tolti dalla coda e quindi se ci fosse il 3 giocatore diventa 1 nella coda.**

```
    if(listaplayer.size()%2 == 0){  
        String idPartita = readyForGame();  
        System.out.println("after ready"+idPartita);  
        return idPartita;  
    }  
}
```

**In caso se viene restituita la stringa diversa da “new” significa che siamo il giocatore mod % 2 == 0. Viene chiamata la funzione StartNewGame () nella classe MenuOnline**

```
130     public void startNewGame() {  
131         @SuppressLint("HardwareIds") String mId = Settings.Secure.getString(getApplicationContext(), Settings.Secure.ANDROID_ID);  
132         RequestQueue requestQueue = Volley.newRequestQueue(context: this);  
133         String url1 = url + "start";  
134         Uri.Builder builder = Uri.parse(url1).buildUpon();  
135         builder.appendQueryParameter("idp", mId);  
136         //builder.appendQueryParameter("param2", "value2");  
137         String urlWithParams = builder.build().toString();  
138  
139  
140         StringRequest stringRequest = new StringRequest(  
141             Request.Method.GET, urlWithParams,  
142             new Response.Listener<String>() {  
143                 no usages  
144                 @Override  
145                 public void onResponse(String response) {  
146                     // Gestisci la risposta come stringa  
147                     if(!response.contentEquals("wait")) {  
148                         idPartita = (response);  
149  
150                         messageOnDisplay.setText("Inizia il gioco");  
151                         Intent i = new Intent(packageContext: MenuOnline2.this, Multiplayer.class);  
152                         i.putExtra(name: "idPlayer", idPlayer);  
153                         i.putExtra(name: "idPartita", idPartita);  
154                         finish(); //Kill the activity from which you will go to next activity  
155                         startActivity(i);  
156                 }  
157             }  
158         }  
159     }  
160     @Override  
161     public void onProgressUpdate(Integer progress) {  
162         super.onProgressUpdate(progress);  
163         Log.d(tag: "progress", msg: "Progress: " + progress);  
164     }  
165     @Override  
166     public void onErrorResponse(VolleyError error) {  
167         super.onErrorResponse(error);  
168         messageOnDisplay.setText("Error: " + error.getMessage());  
169     }  
170     @Override  
171     public void onFinish() {  
172         super.onFinish();  
173         Log.d(tag: "onFinish", msg: "Finish");  
174     }  
175     requestQueue.add(stringRequest);  
176 }  
177 }
```

```
154             startActivity(i);  
155         }  
156     }  
157     else {  
158         messageOnDisplay.setText("Aspetta un avversario");  
159         new Handler().postDelayed(new Runnable() {  
160             @Override  
161             public void run() {  
162                 Log.d(tag: "pausa", msg: "waiting new game");  
163                 startNewGame();  
164             }  
165         }, delayMillis: 5000); // Ritardo di 5 secondi  
166     }  
167 },  
168 new Response.ErrorListener() {  
169     no usages  
170     @Override  
171     public void onErrorResponse(VolleyError error) {  
172         // Gestisci eventuali errori  
173         Log.e(tag: "VolleyError", msg: "Errore: " + error.getMessage());  
174     }  
175     requestQueue.add(stringRequest);  
176 }  
177 }
```

**Nella funzione StartNewGame () viene fatta la chiamata alla API per ottenere id della partita il quale viene generato da unione di due id dei giocatori della partita. In chiamate future alla nostra API sarà indispensabile fornire ID partita perché è memorizzata nella Lista. Dopo che ha ricevuto ID partita lo mette nelle putExtra di intent e lancia la nuova activity che è activity di gioco Multiplayer**

Le classi principali sono

## Card

- Card: composto da due interi e due array di Stringhe

```
1 package com.tura.test.cardsStuff;
2
3     18 usages
4     public class Card {
5         3 usages
6         private int rango,seme;
7         1 usage
8         private static String[] semi = {"Denaro", "Spada", "Bastone", "Coppa"};
9         3 usages
10        private static String[] ranghi= {"Asso","2","3","4","5","6","7","fante","cavallo","re"};
11
12        no usages
13        >     public static String rangoAsString( int __rank ) { return ranghi[__rank]; }
14
15        no usages
16        >     public static String semiAsString( int __rank ) { return semi[__rank]; }
17
18        2 usages
19        Card(int seme, int rango){
20            this.rango=rango;
21            this.seme=seme;
22        }
23    }
```

**funzioni: per recuperare il rango o il seme della carta**

```
22 |
23
24 >     6 usages
24   >     public int getRango() { return rango; }
27
28 >     6 usages
28   >     public int getSeme() { return seme; }
31
32 >     4 usages
32   >     public String getValueRango(int v) { return ranghi[v]; }
35
36 >     4 usages
36   >     public String getValueSeme(int n) { return semi[n]; }
39
40     @Override
41 ⌂     public String toString() {
42       return "Card{" +
43           "rango=" + rango +
44           ", seme=" + seme +
45           '}';
46     }
47 }
```

## Deck

- **Deck:** composto da un ArrayList , è stato utilizzato un ArrayList quando logicamente la prima cosa da usare un Array. La prima idea di implementazione e gestione delle carte fu quella di eliminare gli elementi usati e quindi ridimensionare array per non usare indici... Consiglio è di utilizzare indici e Array
- Idea è di creare array ordinato con i due FOR (riga 4-6) dove prima viene scelto il seme da un elenco prefissato prima nella classe Card, al prossimo FOR viene mescolato il nostro array ordinato in modo tale che gli elementi non sono più ordinati.
- *Tips: è possibile creare direttamente un array che abbiamo in output con solo FOR*

```
6 public class Deck {  
7     13 usages  
8     private ArrayList<Card> cards;  
9     3 usages  
10    int index_1, index_2;  
11    2 usages  
12    Card temp;  
13    public Deck(){  
14        cards = new ArrayList<Card>();  
15        Random generator = new Random();  
16  
17        for (int a=0; a<4; a++)  
18        {  
19            for (int b=0; b<10; b++)  
20            {  
21                cards.add( new Card(a,b) );  
22            }  
23        }  
24  
25        int size = cards.size();  
26  
27        for (int i=0; i<100; i++)  
28        {  
29            index_1 = generator.nextInt( size );  
30            index_2 = generator.nextInt( size );  
31  
32            temp = (Card) cards.get( index_2 );  
33            cards.remove( index_2 );  
34            cards.add( index_1, temp );  
35        }  
36    }  
37  
38    public void printDeck()  
39    {  
40        for (Card c : cards)  
41        {  
42            System.out.println(c);  
43        }  
44    }  
45}
```

- Infine abbiamo due funzioni , una delle due ha proprio la funzione per togliere una carta dal mazzo

```

22     int size = cards.size();
23
24     for (int i=0; i<100; i++)
25     {
26         index_1 = generator.nextInt( size );
27         index_2 = generator.nextInt( size );
28
29         temp = (Card) cards.get( index_2 );
30         cards.set( index_2 , cards.get( index_1 ) );
31         cards.set( index_1, temp );
32     }
33 }
34 public Deck(boolean v){
35     cards = new ArrayList<Card>();
36
37     for (int a=0; a<4; a++)
38     {
39         for (int b=0; b<10; b++)
40         {
41             cards.add( new Card(a,b) );
42         }
43     }
44
45     int size = cards.size();
46
47 }
48
49 > 1 usage
50 >     public Card drawFromDeck() { return cards.remove( index: cards.size()-1 ); }
51 > no usages
52 >     public int getTotalCard() { return cards.size(); }
53 >
54
55
56

```

## Hand

- Hand: serve per gestire le 3 carte nella mano
- Nel codice di sotto viene creato array cards che serve per contenere i valori per ogni carta nel mazzo , in pratica alla posizione x corrisponde carta di posto x nel array

```
3  public class Hand {  
4      27 usages  
5      private Card[] cards;  
6      8 usages  
7      private int[] value;  
8  
9      4 usages  
10     private int pointerArrayCard = 7;  
11  
12     public Hand(Deck d){  
13         value = new int[40];  
14         cards = new Card[40];  
15         for (int x=0; x<40; x++)  
16         {  
17             cards[x] = d.drawFromDeck(); //  
18             switch (cards[x].getRango()){  
19                 case 0: value[x]=11;break;  
20                 case 2: value[x]=10;break;  
21                 case 9: value[x]=4;break;  
22                 case 8: value[x]=3;break;  
23                 case 7: value[x]=2;break;  
24                 default: value[x]=0;  
25             }  
26         }  
27     }
```

**La funzione NextCard ci serve per prelevare una carta dal mazzo**

**La funzione getCardPos ci restituisce la carta prendendo in input la posizione x nel array delle carte**

**La funzione moddingHand : non è stata usata ma fu progettata per poter sostituire una carta nella posizione X**

```
26     public Card NextCard(){
27         pointerArrayCard++;
28         if(pointerArrayCard == 39) return cards[6];
29         if(pointerArrayCard>39) return null;
30         else return cards[pointerArrayCard-1];
31     }
32
33     no usages
34     public Card NextCardV2(int pos){
35         //pointerArrayCard++;
36         if(pos == 39) return cards[6];
37         if(pos>39) return null;
38         else return cards[pos];
39     }
40
41     3 usages
42     >     public Card getCardPos(int pos){return cards[pos];}
43
44     6 usages
45     >     public void moddingHand(Card income,int pos) {cards[pos] = income;}
46     no usages
47     >     public String CardsOnHandToString(){
48         //cards[0].getValueSeme(cards[0].getSeme())+" "+ cards[1].getValueSeme(cards[1].getSeme())+" "+cards[2].
49         //Log.d("solve1", " "+cards[0].getRango());
50         //Log.d("solve2",""+cards[0].getValueRango(9));
51         //Log.d("Seme1",""+ cards[0].getSeme()+cards[1].getSeme()+cards[2].getSeme());
52         return cards[0].getValueSeme(cards[0].getSeme())+" "+cards[0].getValueRango(cards[0].getRango())+
53             " "+ cards[1].getValueSeme(cards[1].getSeme())+" "+cards[1].getValueRango(cards[1].getRango())+
54             " "+cards[2].getValueSeme(cards[2].getSeme())+" "+cards[2].getValueRango(cards[2].getRango());
```

Poi seguono varie funzioni per il recupero delle informazioni come il seme o rango

```
55     public String CardOnHandOneToOne(int pos){  
56         return cards[pos].getValueSeme(cards[pos].getSeme())+" "+cards[pos].getValueRango(cards[pos].getRango());  
57     }  
58  
59     public String getInfoCard1(int pos) {  
60         int x = cards[pos].getSeme();  
61         switch (x) {  
62             case 0:  
63                 return "d";  
64             case 1:  
65                 return "s";  
66             case 2:  
67                 return "b";  
68             default:  
69                 return "c";  
70         }  
71     }  
72  
73     no usages  
74     public String[] getInfoCardRangoSeme(int pos){  
75         String[] array = new String[2];  
76         String res1,res2;  
77         res1 = getInfoCard1(pos);  
78         res2 = String.valueOf(getInfoCard2(pos));  
79         array[0]= res1;  
80         array[1]=res2;  
81         return array;  
82     }  
83 >     public int getInfoCard2(int pos) { return cards[pos].getRango(); }  
84
```

## SinglePlayerActivity

La classe per la gestione di giocatore singolo SinglePlayerActivity

- Crea gli oggetti Card , Deck, Hand . Poi viene creata interfaccia utente da gioco, per poi procedere con definizione della partita. Una volta cominciata la partita ci sono le funzioni che controllano e gestiscono le carte sia nella mano del giocatore sia nella mano dell'avversario, oltre a gestire il confronto tra le due carte e applicazione delle regole del gioco

```
54      @Override
55  ↗      protected void onCreate(Bundle savedInstanceState) {
56          super.onCreate(savedInstanceState);
57          setContentView(R.layout.game_screen_test);
58
59          DisplayMetrics displayMetrics = new DisplayMetrics();
60          getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);
61          height = displayMetrics.heightPixels;
62          width = displayMetrics.widthPixels;
63
64          imageView = findViewById(R.id.im7);
65
66          tx1 = findViewById(R.id.im1);
67          tx2 = findViewById(R.id.im2);
68          tx3 = findViewById(R.id.im3);
69          t0 = findViewById(R.id.im4);
70          t1 = findViewById(R.id.im5);
71          t2 = findViewById(R.id.im6);
72          carta1 = findViewById(R.id.carta1);
73          carta2 = findViewById(R.id.carta2);
74          top = findViewById(R.id.im11);
75          testoSopra = findViewById(R.id.textView);
76          testogiu = findViewById(R.id.textView2);
77          ResultText = findViewById(R.id.resultText);
78          ng = findViewById(R.id.ng);
79          menuB = findViewById(R.id.menuButton);
80          FrameTop = findViewById(R.id.coinWithTextUp);
81          FrameDown = findViewById(R.id.coinWithTextDown);
82          coinUptext = findViewById(R.id.coinText);
83          coinDownText = findViewById(R.id.coinTextDown);
84          SemePrincipale = findViewById(R.id.semePrincipale);
```

Nel metodo onCreate() vengono letti i parametri del display e assegnamento dei riferimenti agli oggetti , dove tx1,tx2,tx3 sono carte di sopra, mentre t0,t1,t2 sono le carte di sotto , poi ci sono carta 1 e carta 2 che rappresentano le carte in mezzo carta 1 quella di sinistra mentre carta 2 di destra. “top” rappresenta la carta in mezzo a sinistra che rappresenta il seme principale.

```
90     Deck D = new Deck();
91     Context c = getApplicationContext();
92     Hand hand = new Hand(D);
93     //      for(int i = 0; i<40; i++){
94     //          Log.d("mazzo",""+hand.CardOnHandOneToOne(i)+"\n");
95     //      }
96
97     //per ottenere il nome della risorsa
98     String s0 = "drawable/" + hand.getInfoCard1(x1)+hand.getInfoCard2(x1);
99     String s1 = "drawable/" + hand.getInfoCard1(x2)+hand.getInfoCard2(x2);
100    String s2 = "drawable/" + hand.getInfoCard1(x3)+hand.getInfoCard2(x3);
101
102    //      String s3 = "drawable/" + hand.getInfoCard1(y1)+hand.getInfoCard2(y1);
103    //      String s4 = "drawable/" + hand.getInfoCard1(y2)+hand.getInfoCard2(y2);
104    //      String s5 = "drawable/" + hand.getInfoCard1(y3)+hand.getInfoCard2(y3);
105
106    String s6 = "drawable/" + hand.getInfoCard1(z)+hand.getInfoCard2(z);
107
108
109    //id della risorsa
110    int id0 = c.getResources().getIdentifier(s0, defType: null,c.getPackageName());
111    int id1 = c.getResources().getIdentifier(s1, defType: null,c.getPackageName());
112    int id2 = c.getResources().getIdentifier(s2, defType: null,c.getPackageName());
113
114
115    int id6 = c.getResources().getIdentifier(s6, defType: null,c.getPackageName());
116
117    t0.setImageResource(id0);
118    t1.setImageResource(id1);
119    t2.setImageResource(id2);
120
```

Per impostare immagine dinamicamente di un ImageView serve id della risorsa, dobbiamo prima definire una stringa che è composta da PATH per la risorsa e poi creare un intero che rappresenta un numero di riferimento a quella risorsa.

```
117         t0.setImageResource(id0);
118         t1.setImageResource(id1);
119         t2.setImageResource(id2);
120
121
122         imageView.setImageResource(id6);
123
124
125         //imposto dimensioni dei vari elementi
126         t0.setLayoutParams().width = width/4;
127         t0.setLayoutParams().height = height/5;
128         t1.setLayoutParams().height = height/5;
129         t1.setLayoutParams().width = width/4;
130         t2.setLayoutParams().height = height/5;
131         t2.setLayoutParams().width = width/4;
132
133         tx2.setLayoutParams().height = height/5;
134         tx2.setLayoutParams().width = width/4;
135         tx1.setLayoutParams().height = height/5;
136         tx1.setLayoutParams().width = width/4;
137         tx3.setLayoutParams().height = height/5;
138         tx3.setLayoutParams().width = width/4;
139
140
141         imageView.setLayoutParams().width = width/3;
142         imageView.setLayoutParams().height = height/6;
143
144         top.setLayoutParams().width = width/3;
145         top.setLayoutParams().height = height/10;
146         top.setTranslationY(height/22);
147         top.setImageAlpha(255);
```

Imposto dinamicamente le dimensioni delle carte , considerando che le carte sono 3 su e giù non ci basta dividere la larghezza dello schermo per 3 , quindi dobbiamo dividere per 4. Quindi a ogni carta è assegnato un valore di larghezza pari ad un  $\frac{1}{4}$ . Per altezza ho scelto 1/5

```

180         t0.setOnClickListener(new View.OnClickListener() {
181             @Override
182             ↗ public void onClick(View view) {
183
184                 if(carte<39) {
185                     Log.d( tag: "valori2", msg: "valori "+carte+" "+x1+" "+y1+" "+y2+" "+y3);
186
187                     setClickable(t0, x: width/2+width/14, y: -height/3, top: false, id: 1, last: false);
188
189                     int AI = AI(hand,x1,y1,y2,y3);           //viene scelta migliore carta e restituito il suo indice
190
191                     tx = pcMove();                         //viene scelta a caso animazione carta

```

Sul ImageView viene applicato il metodo listener che permette intercettare il click su ImageView (la nostra carta ,una delle tre).

Poi faccio il controllo perché nella gestione ho suddiviso in due casi , la prima è quando siamo con qualsiasi mano diversa da ultima e poi ultima mano che ha bisogno di gestione particolare.

Quindi faccio il controllo sulla variabile carte , se non è ultima mano allora chiamo la funzione setClickable ()

```

930     public void setClickable(ImageView im,int x,int y,boolean top,int id,boolean last){
931
932         ObjectAnimator animX = ObjectAnimator.ofFloat(im,   propertyName: "translationX", x);
933         ObjectAnimator animY = ObjectAnimator.ofFloat(im,   propertyName: "translationY", y);
934
935         // Combina le animazioni
936         AnimatorSet animatorSet = new AnimatorSet();
937         animatorSet.playTogether(animX, animY);
938         animatorSet.setDuration(500); // Durata in millisecondi
939         animatorSet.setInterpolator(new AccelerateDecelerateInterpolator());
940         if(top) flipCard(im,id);
941         animatorSet.start();
942
943         new Handler().postDelayed(new Runnable() {
944             @Override
945             ↑ public void run() {
946                 im.setVisibility(View.INVISIBLE);
947                 animatorSet.reverse();
948                 if(top) flipCard(im,r2);
949             }
950         }, delayMillis: 3500);
951         new Handler().postDelayed(new Runnable() {
952             @Override
953             ↑ public void run() {
954                 //testoGiù.setVisibility(View.INVISIBLE);
955                 //testoSopra.setVisibility(View.INVISIBLE);
956                 if(!last) im.setVisibility(View.VISIBLE);
957             }
958         }, delayMillis: 4000);

```

, che ha funzione di animare il movimento della carta.

Con new Handler().postDelayed possiamo eseguire delle istruzioni con un ritardo di x millesekondi. Una volta che viene effettuata l'animazione dobbiamo **tornare**

indietro, altrimenti la oggetto (carta) riamane alla nuova posizione , per questo che facciamo `animatorSet.reverse();` , poi in caso di una carta da sopra dobbiamo che girare la carta con la funzione flipCard ()

```
1067     public void flipCard(ImageView tt,int id){  
1068         @SuppressLint("ResourceType") AnimatorSet flipOut = (AnimatorSet) AnimatorInflater.loadAnimator(context, R.anim.flip_out);  
1069         @SuppressLint("ResourceType") AnimatorSet flipIn = (AnimatorSet) AnimatorInflater.loadAnimator(context, R.anim.flip_in);  
1070  
1071         flipOut.setTarget(tt);  
1072         flipOut.start();  
1073  
1074         flipOut.addListener(new Animator.AnimatorListener(){  
1075             @Override  
1076             public void onAnimationEnd(Animator animation){  
1077                 // Cambia immagine quando la carta è invisibile (metà del flip)  
1078                 tt.setImageResource(id); // Mostra la maglia  
1079  
1080  
1081                 isBackVisible = !isBackVisible; // Inverte stato  
1082  
1083                 // Avvia la seconda parte dell'animazione  
1084                 flipIn.setTarget(tt);  
1085                 flipIn.start();  
1086             }  
1087  
1088             @Override  
1089             public void onAnimationStart(Animator animation) {}  
1090             @Override  
1091             public void onAnimationCancel(Animator animation) {}  
1092             no usages  
1093             @Override  
1094             public void onAnimationRepeat(Animator animation) {}  
1095         });  
1096     };
```

---

Dopo setClickable abbiamo la funzione Al(hand,x1,y1,y2,y3); che prende in input riferimento al mazzo con hand , x1 carta del giocatore , e le tre carte del avversario. La funzione serve per individuare tra le tre carte la carta migliore da dare in risposta alla nostra carta.

Il primo controllo che facciamo è quello per vedere se la carta scelta da noi vale o meno dei punti , se non vale niente allora cerchiamo anche noi di dare una carta di valore nullo perché, secondo le regole del gioco se non avversario non gioca la carta dello stesso seme o di seme della mano perde, quindi è più probabile che abbiamo una carta che ha valore nullo e se va bene ha lo stesso seme , ad ogni modo carte di valore nullo non fanno la differenza.

Nelle immagine di sotto è presente il codice

```

815     @v    private int AI(Hand h, int p1, int p2, int p3, int p4) {
816         int v = h.getValue(p2) ;
817         String seme1 = h.getInfoCard1(p1), MainSeme = h.getInfoCard1( pos: 6);
818
819         String carta1,carta2,carta3;
820         carta1 = h.CardOnHandOneToOne(p2);
821         carta2 = h.CardOnHandOneToOne(p3);
822         carta3 = h.CardOnHandOneToOne(p4);
823
824         /*dove fare in modo che se la carta di PC e' uscita allora non la deve usare piu
825         quindi la mia strategia assegnare 0 alla carta uscita quindi p2 , p3 o p4 = 0 e al loro posto
826         assegno una delle p2 , p3 o p4
827         */
828         for(int i = 0; i<3 ; i++){
829             if(p2 == 0) p2 = p3;
830             if(p3 == 0) p3 = p4;
831             if(p4 == 0) p4 = p2;
832         }
833         if(h.getValue(p1) == 0 ) { //se carta del giocatore 1 e' da 0 punti cerchiamo anche noi di dare una carta nulla
834             if((v >= h.getValue(p3))&& (h.getInfoCard1(p3).compareTo(MainSeme) != 0)) {return p3;}
835             if((v >= h.getValue(p4))&& (h.getInfoCard1(p4).compareTo(MainSeme) != 0)){return p4;}
836             else return p2;
837         }
838
839         /*
840         carta del giocatore 1 ha un valore allora dobbiamo capire se tra le nostre 3 carte c'e' una carta
841         dello stesso seme ma di rango maggiore oppure una carta cheha lo stesso seme della carta scoperta sul tavolo
842         */
843

```

Poi passiamo al caso quando la carta del giocatore ha un valore, allora dobbiamo scegliere una carta che vince sulla carta del giocatore , in caso contrario scegliere una carta di valore nullo per minimizzare il numero di punti guadagnati dall'avversario.

```

844             else {//carta del giocatore ha valore
845                 if (seme1.compareTo(h.getInfoCard1(p2)) == 0) {
846                     if (h.getValue(p1) < h.getValue(p2)) return p2;
847                     else {
848                         if(h.getValue(p1) < h.getValue(p3)) return p3;
849                         else return p4;
850                     }
851                 }
852
853                 /*
854                 Se il giocatore ha la carta dello stesso seme con carta principale allora dobbiamo
855                 vedere se e' una carta che ha valore asso ,tre , re ,cavalllo o fante altrimenti se la
856                 carta non ha un valore dobbiamo restituire una carta senza un valore o di alore piu
857                 piccolo possibile
858
859                 */
860
861                 else if (seme1.compareTo(h.getInfoCard1( pos: 6))== 0) {
862                     if(seme1.compareTo(h.getInfoCard1(p2)) == 0) {
863                         if((v>0)&& (v<h.getValue(p2))) return p2;
864                     }
865                     else if(seme1.compareTo(h.getInfoCard1(p3)) == 0) {
866                         if((v>0)&& (v<h.getValue(p3))) return p3;
867                     }
868                     else if(seme1.compareTo(h.getInfoCard1(p4)) == 0) {
869                         if((v>0)&& (v<h.getValue(p4))) return p4;
870                     }
871                 }
872
873                 //prima carta e' dello stesso seme di carta principale?
874

```

```

872
873     //prima carta e' dello stesso seme di carta principale?
874
875     else if (h.getInfoCard1(p2).compareTo(h.getInfoCard1( pos: 6)) == 0) return p2;
876     else if (h.getInfoCard1(p3).compareTo(h.getInfoCard1( pos: 6)) == 0) return p3;
877     else if (h.getInfoCard1(p4).compareTo(h.getInfoCard1( pos: 6)) == 0) return p4;
878
879         //cerchiamo il valore piu piccolo
880     else if ((h.getValue(p2) <= h.getValue(p3)) && (h.getValue(p2) <= h.getValue(p4)))
881         return p2;
882     else if ((h.getValue(p3) <= h.getValue(p2)) && (h.getValue(p3) <= h.getValue(p4)))
883         return p3;
884     else return p4;
885
886
887 }
888
889     return p2;

```

Tornando a SinglePlayer class:

```

192
193     tx = pcMove();           //viene scelta a caso carta per animazione
194
195     t0.setEnabled(false);
196     t1.setEnabled(false);
197     t2.setEnabled(false);
198
199     String lstS = "drawable/"+hand.getInfoCard1(x1)+hand.getInfoCard2(x1);
200     int lstI = c.getResources().getIdentifier(lstS, defType: null,c.getPackageName());
201     carta1.setImageResource(lstI);
202
203     String sc = "drawable/"+hand.getInfoCard1(AI)+hand.getInfoCard2(AI);    //creo la stringa a cui assegno
204     int idAI = c.getResources().getIdentifier(sc, defType: null,c.getPackageName()); //intero
205     setClickable(tx1, x: width/4, y: height/3, top: true,idAI, last: false);      //carta2.setImageResource(idAI);          //imposto carta PC pick

```

Dopo che è stata scelta la carta da giocare lato AI viene assegnata a una delle 3 carte la nuova immagine che si riferisce alla carta scelta. X1 è indice che nella prima mano vale 0 e viene incrementato in modo intelligente ogni volta che la prima carta viene scelta.

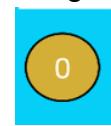
```

206
207     if(compareA(hand,z,x1,AI) == 1) { //punteggioA - del giocatore
208         new Handler().postDelayed(new Runnable() {
209             @Override
210             public void run() {
211                 goingDown(tx1);
212                 goingDown(t0);
213                 punteggioA = punteggioA+hand.getValue(x1)+hand.getValue(AI); //comparo e assegno i punteggi
214                 testoGiu.setText(String.valueOf( hand.getValue(x1)+hand.getValue(AI)));
215                 CoinsAdd(CoinDown, pos: 1);
216                 updateCoinValue(coinDownText , punteggioA);
217                 //backCardToPosition(tx1);
218                 //backCardToPosition(t0);
219             }
220         }, delayMillis: 1);
221     }

```

If controlla se la funzione compareA che prende in input riferimento al mazzo, indice di carta che ha il seme di gioco , e poi le due carte : carta del giocatore e carta scelta da AI. In base alla risposta 1 o 2 viene seguito if or else. goingDown(tx1) sono delle funzioni che servono per muovere le carte. Punteggio A è il punteggio che appartiene al giocatore , subito dopo vediamo funzioni CoinsAdd e updateCoinValue

La prima serve per far apparire la monetina e la muove con animazione verso la monetina già esistente , quando passa sopra viene eseguita updateConValue che



aggiorna il valore dei punti (il testo nella monetina)

```
222     else{
223         new Handler().postDelayed(new Runnable() {
224             @Override
225             public void run() {
226                 goingTop(t0);
227                 goingTop(tx1);
228                 punteggioB = punteggioB+hand.getValue(x1)+hand.getValue(AI);
229                 testoSopra.setText(String.valueOf( hand.getValue(x1)+hand.getValue(AI)));
230                 CoinsAdd(CoinUp, pos: 1);
231                 updateCoinValue(coinUptext, punteggioB);
232
233                 //backCardToPosition(tx1);
234                 //backCardToPosition(t0);
235             }
236         }, delayMillis: 1);
237     }
238 }
```

Viene seguito if in caso se vince la carta del utente , mentre else in caso di carta vincente di AI

---

In seguito è riportato il codice per compareA():

```
736     @ private int compareA(Hand h, int a, int b, int c){
737         Log.d( tag: "valori", msg: "carta giocatore "+h.getValue(b)+h.getInfoCard1(b)+h.getInfoCar
738         //a posizione carta con seme della partita , b carta del giocatore, c carta AI
739         String semePartita = h.getInfoCard1(a);
740         String cartaP1 = h.getInfoCard1(b);
741         String cartaP2 = h.getInfoCard1(c);
742
743         // Controlla se le carte 2 e 3 appartengono al seme della partita
744         boolean carta2SegueSeme = cartaP1.contentEquals(semePartita);
745         boolean carta3SegueSeme = cartaP2.contentEquals(semePartita);
746
747         // Confronto delle carte secondo le regole
748         if (carta2SegueSeme && !carta3SegueSeme) {
749             return 1;
750         } else if (!carta2SegueSeme && carta3SegueSeme) {
751             return 2;
752         } else if (carta2SegueSeme && carta3SegueSeme) {
753             // Se entrambe appartengono al seme della partita, vince quella con rango maggiore
754             return h.getInfoCard2(b) > h.getInfoCard2(c) ? 1 : 2;
755         } else {
756             // Se nessuna segue il seme della partita, vince la carta con rango maggiore
757             if(h.getInfoCard1(b).contentEquals(h.getInfoCard1(c)))
758                 return h.getInfoCard2(b) > h.getInfoCard2(c) ? 1 : 2;
759             else return 1;
760         }
761     }
```

Torniamo di nuovo alla classe SinglePlayer, seguente codice viene eseguito dopo if/else citati sopra , vengono aggiornati indici di carte AI con la funzione NextPos()

```
240         t0.postDelayed(new Runnable() {
241             @Override
242             public void run() {
243                 //t0.setImageAlpha(255);
244                 testoSopra.setText("");
245                 testoGiu.setText("");
246                 String pc1="";
247                 x1 = NextPos(x1);
248                 if(AI == y1) {
249                     y1= NextPosPC(y1);
250
251                 }
252                 if(AI == y2){
253                     y2= NextPosPC(y2);
254
255                 }
256                 if(AI == y3){
257                     y3= NextPosPC(y3);
258                 }
259
260
261                 //hand.moddingHand(g1,x1);           //assegna in posizione 0 la carta g1
262
263                 String s0 = "drawable/"+hand.getInfoCard1(x1)+hand.getInfoCard2(x1);
264
265                 int id0 = c.getResources().getIdentifier(s0, defType: null,c.getPackageName());
266             }
267         }
```

NextPos():

```
722     private int NextPos (int pos){
723         if(secondo){
724             even = 7;
725             secondo = false;
726             return even;
727         }
728         else{
729             even = even+2;
730             carte = even;
731             return even;
732         }
733     }
734 }
```

NextPosPC():

```
710     private int NextPosPC (int pos){  
711         if(odd == 38) return 6;  
712         if(primo){  
713             odd = 8;  
714             primo = false;  
715             return odd;  
716         }  
717         else{  
718             odd = odd+2;  
719             return odd;  
720         }  
721     }
```

Continua codice SinglePlayer

```
263             String s0 = "drawable/" + hand.getInfoCard1(x1) + hand.getInfoCard2(x1);  
264  
265             int id0 = c.getResources().getIdentifier(s0, defType: null, c.getPackageName());  
266  
267             t0.setImageResource(id0);  
268  
269             //modifico carta pc  
270  
271  
272  
273  
274  
275             tx.setImageAlpha(255);  
276             if(x1>37){  
277                 top.setImageResource(id0);  
278                 imageView.setImageResource(id0);  
279             }  
280  
281             t0.setEnabled(true);  
282             t1.setEnabled(true);  
283             t2.setEnabled(true);  
284         }  
285     }, delayMillis: 4000);  
286 }
```

Viene nascosta la carta in mezzo tx e vengono abilitate le carte del giocatore t0 ,t1, t2.

---

In seguito viene riportato il codice in caso di ultima mano

```

289     else { //blocco dell'ultima mano
290         Log.d( tag: "valori2", msg: "valori "+x1+" "+y1+" "+y2+" "+y3);
291         //x1 = 39;
292         top.setImageAlpha(0);
293
294         int AI = AI(hand,x1,y1,y2,y3);
295
296         String lstS = "drawable/"+hand.getInfoCard1(x1)+hand.getInfoCard2(x1);
297         int lstI = c.getResources().getIdentifier(lstS,null,c.getPackageName());
298         carta1.setImageResource(lstI);
299         String sc = "drawable/"+hand.getInfoCard1(AI)+hand.getInfoCard2(AI);
300         int idAI = c.getResources().getIdentifier(sc, defType: null,c.getPackageName());
301         carta2.setImageResource(idAI);
302         carta1.setImageAlpha(255);
303         carta2.setImageAlpha(255);
304         setClickable(t0, x: width/2+width/14, y: -height/3, top: false, id: 1, last: true);
305         setClickable(tx1, x: width/4, y: height/3, top: true,idAI, last: true);
306
307         if(compareA(hand,z,x1,AI) == 1) {
308             goingDown(tx1);
309             goingDown(t0);
310             punteggioA = punteggioA+hand.getValue(x1)+hand.getValue(AI);    //comparo e assegno i punteggi
311             testoGiu.setText(String.valueOf( hand.getValue(x1)+hand.getValue(AI)));
312             CoinsAdd(CoinDown, pos: 1);
313             updateCoinValue(coinDownText , punteggioA);
314         }
315         else{
316             goingTop(t0);
317             goingTop(tx1);
318             punteggioB = punteggioB+hand.getValue(x1)+hand.getValue(AI);
319             testoSopra.setText(String.valueOf( hand.getValue(x1)+hand.getValue(AI)));
320             CoinsAdd(CoinUp, pos: 1);
321             updateCoinValue(coinUptext , punteggioB);
322         }
323         if(y1 == AI) {
324             y1 = 0;
325
326         }
327         if(y2 == AI){
328             y2 = 0;
329
330         }
331         if(y3 == AI){
332             y3 = 0;
333
334         }
335         tx1.setVisibility(View.INVISIBLE);
336         t0.setVisibility(View.INVISIBLE);
337         imageView.setImageAlpha(0);
338         //tx.setImageAlpha(0);
339         //t0.setEnabled(false);
340
341         if((y1==0)&&(y2==0)&&(y3==0))newGame(lose,punteggioB,punteggioA);
342
343     }

```

Il codice è molto simile a quello di prima , ma ci sono delle differenze come ad esempio la gestione dei indici , oltre alla funzione newGame che toglie tutte le carte in caso se tutti indici sono uguali a 0 allora la partita è conclusa ed è possibile visualizzare il menu chiamando la funzione newGame

```
892     private void newGame(boolean result,int pc ,int player){
893         RelativeLayout layout = (RelativeLayout) findViewById(R.id.layoutR);
894         if(result) layout.setBackgroundColor(getColor(R.color.greenVictory));
895         else layout.setBackgroundColor(getColor(R.color.redLose));
896         t2.setVisibility(View.INVISIBLE);
897         tx1.setVisibility(View.INVISIBLE);
898         tx2.setVisibility(View.INVISIBLE);
899         tx3.setVisibility(View.INVISIBLE);
900         t0.setVisibility(View.INVISIBLE);
901         t1.setVisibility(View.INVISIBLE);
902         testoSopra.setText("");
903         testoGiu.setText("");
904         FrameTop.setVisibility(View.INVISIBLE);
905         FrameDown.setVisibility(View.INVISIBLE);
906         ng.setVisibility(View.VISIBLE);
907         menuB.setVisibility(View.VISIBLE);
908         ResultText.setVisibility(View.VISIBLE);
909         ResultText.setText("Punteggio:\n pc "+pc+"\n giocatore "+player);
910         ng.setOnClickListener(new View.OnClickListener() {
911             @Override
912             ↗ public void onClick(View view) {
913                 Intent intent = getIntent();
914                 finish();
915                 startActivityForResult(intent);
916             });
917             menuB.setOnClickListener(new View.OnClickListener() {
918                 @Override
919                 ↗ public void onClick(View view) {
920                     Intent i = new Intent(packageContext: SinglePlayerActivity2.this, MenuPrincipale.class);
921                     finish(); //Kill the activity from which you will go to next activity
922     }
```

## Multiplayer

### La classe per la gestione di giocatore multiplayer Multiplayer

- Oltre alle funzioni citate sopra nella classe **SinglePlayerActivity** contiene anche funzioni di comunicazione con il server per lo scambio dei messaggi. Lo scambio dei messaggi avviene grazie a JSON e le librerie legati JSON

```
73
74    13 usages
75    RequestQueue requestQueue; // This is our requests queue to process our HTTP requests.
76
77    1 usage
78    String httpLink = "http://";
79    //String ip = "192.168.1.134";
80    1 usage
81    String ip = "10.0.2.2";
82    5 usages
83    String url = httpLink + ip + ":8080/post/";
84
```

Prima di tutto serve definire RequestQueue ossia coda delle richieste che facciamo verso la nostra API. Poi ci conviene impostare indirizzo , dove la parte di http:// è fissa , mentre ip cambia in base alle nostre necessita , in questo caso utilizziamo API in locale quindi ci serve localhost . Dalla documentazione di Android Studio

#### Spazio degli indirizzi di rete

Ogni istanza dell'emulatore viene eseguita dietro un router virtuale o un servizio firewall che lo isola dalle interfacce e dalle impostazioni di rete della tua macchina di sviluppo e da Internet. Un dispositivo emulato non può rilevare la tua macchina di sviluppo o altre istanze dell'emulatore sulla rete. Rileva solo che è connesso tramite Ethernet a un router o firewall.

Il router virtuale per ogni istanza gestisce lo spazio di indirizzamento di rete 10.0.2/24. Tutti gli indirizzi gestiti dal router sono nel formato 10.0.2.**xx**, Dove **xx** è un numero. Gli indirizzi all'interno di questo spazio sono pre-assegnati dall'emulatore o dal router come segue:

Indirizzo di rete	Descrizione
10.0.2.1	Indirizzo router o gateway
10.0.2.2	Alias speciale per l'interfaccia loopback dell'host (127.0.0.1 sulla macchina di sviluppo)

quindi come IP scegliamo 10.0.2.2 e poi c'è la porta 8080 con endpoint che ha come radice post.

#### Regola generale

- Usa GET per leggere dati.
- Usa POST per creare/modificare dati.

## Poi viene eseguita la funzione GetPartita

```
141     //ottieniCarte(idPartita,c); //ottengo le carte dal server
142
143     //per ottenere il nome della risorsa
144     GetPartita(idPlayer,idPartita,hand,c);
145     //    for (int i = 0; i < 40; i++) {
146     //        Log.d("carte",numbers[i].toString());
147     //    }
148
149     new Handler().postDelayed(new Runnable() { //aspetto 1s prima di chiedere al server le carte
150         @Override
151         public void run() {
152             String s0 , s1, s2 ,s6 , t0 ,t1,t2;
153
154             s6 = "drawable/" + hand.getInfoCard1(numbers[39]) + hand.getInfoCard2(numbers[39]);
155         }
156     }, 1000);
157 }
```

```
542     private void GetPartita(String idplay, String idPartita,Hand hand,Context c) {
543         requestQueue = Volley.newRequestQueue(c);
544         String url1 = url + "getPartita";
545         Log.d( tag: "VolleyError", msg: "prima di JSON "+url1);
546         JSONObject postData = new JSONObject();
547         try {
548             postData.put( name: "idPartita", idPartita);
549             postData.put( name: "giocatore", idplay);
550
551         } catch (JSONException e) {
552             e.printStackTrace();
553         }
554         Log.d( tag: "carte", msg: "prima di JSON "+postData.toString());
555         Log.d( tag: "carte", msg: "prima di richiesta "+idPartita);
556         JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(
557             Request.Method.POST, url1,postData,
558             new Response.Listener<JSONObject>() {
559
560                 no usages
561             public void onResponse(JSONObject response) {
562                 if (response == null){
563                     Log.d( tag: "carte", msg: "if null "+idPartita );
564                     handler.postDelayed(() -> GetPartita(idplay, idPartita, hand, c), delayMillis: 1000);
565                 }
566             }
567         });
568 }
```

```
568     else {
569
570         Log.d( tag: "carte", msg: "response != null");
571         // Ottieni il campo "giocatoriExtra" come JSONArray
572         JSONArray carteExtraArray = null;
573         try {
574             myPartita.setIdFirstPlayer(response.getString( name: "idFirstPlayer"));
575             myPartita.setIdSecondPlayer(response.getString( name: "idSecondPlayer"));
576             myPartita.setIdPartita(response.getString( name: "idPartita"));
577             myPartita.setCartaP1(response.getInt( name: "cartaP1"));
578             myPartita.setCartaP2(response.getInt( name: "cartaP2"));
579             myPartita.setCartaP11(response.getInt( name: "cartaP11"));
580             myPartita.setCartaP12(response.getInt( name: "cartaP12"));
581             myPartita.setCartaP13(response.getInt( name: "cartaP13"));
582             myPartita.setCartaP21(response.getInt( name: "cartaP21"));
583             myPartita.setCartaP22(response.getInt( name: "cartaP22"));
584             myPartita.setCartaP23(response.getInt( name: "cartaP23"));

585
586
587             String getIdPlayer = response.getString( name: "turn");
588             carteExtraArray = response.getJSONArray( name: "carte");
589             // Converti JSONArray in una List<String>
590             List<String> carte = new ArrayList<>();
591             //    for (int i = 0; i < carteExtraArray.length(); i++) {
592             //        carte.add(carteExtraArray.getString(i));
593             //        Log.d("carte", " "+carteExtraArray.getString(i));
594             //    }
595             for (int i = 0; i < 40; i++) {
596                 numbers[i] = Integer.parseInt(carteExtraArray.getString(i));
597                 Log.d( tag: "carte", msg: ""+numbers[i]);
598             }
599         }
600     }
601 }
```

```

579     myPartita.setCartaP11(response.getInt( name: "cartaP11"));
580     myPartita.setCartaP12(response.getInt( name: "cartaP12"));
581     myPartita.setCartaP13(response.getInt( name: "cartaP13"));
582     myPartita.setCartaP21(response.getInt( name: "cartaP21"));
583     myPartita.setCartaP22(response.getInt( name: "cartaP22"));
584     myPartita.setCartaP23(response.getInt( name: "cartaP23"));

585
586
587     String getIdPlayer = response.getString( name: "turn");
588     carteExtraArray = response.getJSONArray( name: "carte");
589     // Converti JSONArray in una List<String>
590     List<String> carte = new ArrayList<>();
591     ////
592     ////
593     ////
594     ////
595
596     for (int i = 0; i < carteExtraArray.length(); i++) {
597         carte.add(carteExtraArray.getString(i));
598         Log.d("carte", " "+carteExtraArray.getString(i));
599     }
600     for (int i = 0; i < 40; i++) {
601         numbers[i] = Integer.parseInt(carteExtraArray.getString(i));
602         Log.d( tag: "carte", msg: ""+numbers[i]);
603     }
604     if (idPlayer.contentEquals(getIdPlayer)){
605         turno = true;
606     }
607     else turno = false;

608 } catch (JSONException e) {
609     throw new RuntimeException(e);
610 }

```

Viene fatta la request con POST e si riceve JSONObject un oggetto JSON da cui estraiamo i dati per poi assegnarli alle variabili , a numbers un array di interi è assegnato array di interi che sono stati mescolati sul server , quindi entrambi clienti ricevono lo stesso array mischiato che rappresenta le carte.

**UpdateTurno** serve per aggiornare il turno ed è molto importante in quanto possiamo capire quando è il nostro turno e quindi possiamo giocare la carta

### Riassunto del funzionamento

- 1. Cancella eventuali richieste in sospeso.**
- 2. Crea una nuova richiesta GET per verificare di chi è il turno.**
- 3. Se è il turno del giocatore (response == idplay):**
  - **Imposta turno = true.**
  - **Abilita le carte (enableCarts()).**
  - **Gioca il turno (playTurn(c)).**
- 4. Se non è il turno del giocatore:**
  - **Imposta turno = false.**
  - **Aggiorna le carte (getCarta(idPartita, c)).**

## 5. Ogni 4 secondi la funzione si richiama da sola per verificare di chi è il turno.

```
private void updateTurno(String idplay, String idPartita, Context c){
    requestQueue.cancelAll(request -> true); // Cancella tutte le richieste

    Log.d( tag: "3H", msg: "turno"+turno);
    requestQueue = Volley.newRequestQueue(c);
    String url1 = url + "turno";
    Uri.Builder builder = Uri.parse(url1).buildUpon();
    builder.appendQueryParameter("idp", idPartita);
    String urlWithParams = builder.build().toString();
    StringRequest stringRequest = new StringRequest(
        Request.Method.GET, urlWithParams,
        response -> {
            if(response == null) handler.postDelayed(() -> updateTurno(idplay,idPartita,c), delayMillis: 1000);

            else if(response.contentEquals(idplay)){
                turno = true;
                enableCarts();
                playTurn(c);
            }
            else {
                turno = false;
                getCarta(idPartita,c);
            }
        },
        new Response.ErrorListener() {
            no usages
            @Override
            public void onErrorResponse(VolleyError error) {
                // Gestisci eventuali errori
            }
        }
    );
}
```

**PlayTurn** gestisce il turno del giocatore nell'app di Briscola, permettendogli di selezionare una carta da giocare.

```
490     private void playTurn(Context c){
491         Log.d( tag: "3h", msg: "inside playTurn");
492         if(TurnoFinito){
493             new Handler().postDelayed(new Runnable() {
494                 @Override
495                 public void run() {
496                     // setClickable(Carta1Down,width/2+width/14,-height/3,false,1,false);
497                     // setClickable(Carta2Down,width/4,-height/3,false,1,false);
498                     // setClickable(Carta3Down,0,-height/3,false,1,false);
499                     Carta1Down.setEnabled(true);
500                     Carta2Down.setEnabled(true);
501                     Carta3Down.setEnabled(true);
502                     Carta1Down.setOnClickListener(new View.OnClickListener() {
503                         @Override
504                         public void onClick(View view) {
505                             setCarta(is0, pos: 0);
506                             IoPickCarta = true;
507                             Nostraposizione = 0;
508                             setClickable(Carta1Down, x: width/2+width/14, y: -height/3, top: false, id: 101, last: false);
509                             //setCarta(idPartita,idPlayer,String.valueOf(is0),1,c);
510                         });
511                         Carta2Down.setOnClickListener(new View.OnClickListener() {
512                             @Override
513                             public void onClick(View view) {
514                                 setCarta(is1, pos: 1);
515                                 IoPickCarta = true;
516                                 Nostraposizione = 1;
517                                 setClickable(Carta2Down, x: width/4, y: -height/3, top: false, id: 102, last: false);
518                             });
519                         Carta3Down.setOnClickListener(new View.OnClickListener() {
520                             @Override
521                             public void onClick(View view) {
522                                 setCarta(is2, pos: 2);
523                                 IoPickCarta = true;
524                                 Nostraposizione = 2;
525                                 setClickable(Carta3Down, x: width/4, y: -height/3, top: false, id: 103, last: false);
526                             });
527                         });
528                     });
529                 }
530             });
531         }
532     }
```

- 1. Attende che il turno sia finito.**
- 2. Se il turno è finito, abilita le carte e permette al giocatore di selezionarne una.**
- 3. Se il turno non è finito, riprova dopo 1 secondo.**

**La funzione principale updateScreen. Questa funzione aggiorna periodicamente lo stato del gioco, controllando eventuali nuove mosse degli avversari e aggiornando l'interfaccia utente di conseguenza.**

```

324      private void updateScreen(Context c){
325          handler.postDelayed(() -> updateScreen( c), delayMillis: 5000);
326          String url1 = url + "getAnyUpdate";
327          Uri.Builder builder = Uri.parse(url1).buildUpon();
328          builder.appendQueryParameter("idpartita", idPartita);
329          //builder.appendQueryParameter("param2", "value2");
330          String urlWithParams = builder.build().toString();
331          Log.d( tag: "Carte3", msg: "Updating...");
332          requestQueue = Volley.newRequestQueue(c);
333
334
335          StringRequest stringRequest = new StringRequest(
336              Request.Method.POST, urlWithParams,
337              new Response.Listener<String>() {
338
339              no usages
340              public void onResponse(String response1) {
341                  if (response1 == null){
342                      Log.d( tag: "Carte3", msg: "null...");
343
344                  }
345
346                  else {
347
348                      Log.d( tag: "Carte3", msg: "ho ricevuto i dati...");
349                      try {
350                          JSONObject response = new JSONObject(response1);
351                          String getIdPlayer = response.getString( name: "idFirstPlayer");
352                          int carta , posizione;

```

```

352     int carta , posizione;
353     if(getIdPlayer.contentEquals(idPlayer)){
354         carta= response.getInt( name: "cartaP2");
355         posizione = response.getInt( name: "posCarta2");
356     }
357     else {
358         carta= response.getInt( name: "cartaP1");
359         posizione = response.getInt( name: "posCarta1");
360     }
361     if(carta != -1){

362         if(posizione == 0) CartaPlayer1 = Carta1Top;
363         else if (posizione == 1) CartaPlayer1 = Carta2Top;
364         else CartaPlayer1 = Carta3Top;

365         if(Nostraposizione == 0) CartaPlaye2 = Carta1Down;
366         else if (Nostraposizione == 1) CartaPlaye2 = Carta2Down;
367         else CartaPlaye2 = Carta3Down;

368         cartaAvversario = carta;
369         AvversarioPickCarta = true;
370         if(!IoPickCarta)){
371             Log.d( tag: "alinity" , msg: "carta diversa da -1 e devo mettere la carta" + IoPickCarta + AvversarioPickCarta);
372             enableCarts();
373             playTurn(c);
374         }
375         else{
376             result = compareA(hand,nostraCarta,cartaAvversario);
377             Log.d( tag: "infocarte" , msg: " "+hand.getInfoCard1(nostraCarta)+" "+hand.getInfoCard1(cartaAvversario));
378
379             result = compareA(hand,nostraCarta,cartaAvversario);
380             Log.d( tag: "infocarte" , msg: " "+hand.getInfoCard1(nostraCarta)+" "+hand.getInfoCard1(cartaAvversario));
381             Log.d( tag: "alinity" , msg: "carta diversa da -1 devo dare risultato "+result);
382             if(result == 1) {
383                 goingTop(CartaPlayer1);
384                 goingTop(CartaPlaye2);
385                 punteggioB = punteggioB+hand.getValue(nostraCarta)+hand.getValue(cartaAvversario);
386                 testoSopra.setText(String.valueOf( hand.getValue(nostraCarta)+hand.getValue(cartaAvversario)));
387                 CoinsAdd(CoinUp, pos: 1);
388                 updateCoinValue(coinUptext , punteggioB);
389             }
390             else {
391                 goingDown(CartaPlayer1);
392                 goingDown(CartaPlaye2);
393                 punteggioA = punteggioA+hand.getValue(nostraCarta)+hand.getValue(cartaAvversario); //comparo e ass
394                 testogiu.setText(String.valueOf( hand.getValue(nostraCarta)+hand.getValue(cartaAvversario)));
395                 CoinsAdd(CoinDown, pos: 1);
396                 updateCoinValue(coinDownText , punteggioA);
397             }
398         }
399         setCarta( carta: -1, pos: 0);
400         IoPickCarta = false;
401
402         enableCarts();
403
404     }

405     if(posizione == 0) setClickable(Carta1Top, x: width/4, y: height/3, top: true,carta, last: false);
406     else if (posizione == 1) setClickable(Carta2Top, x: 0, y: height/3, top: true,carta, last: false);
407     else setClickable(Carta3Top, x: -width/3, y: height/3, top: true,carta, last: false);
408
409
410

```

```

407             if(posizione == 0) setClickable(Carta1Top, x: width/4, y: height/3, top: true,carta, [last: false]);
408             else if (posizione == 1) setClickable(Carta2Top, x: 0, y: height/3, top: true,carta, [last: false]);
409             else setClickable(Carta3Top, x: -width/3, y: height/3, top: true,carta, last: false);
410         }
411     }
412   }
413 }
414
415
416
417     } catch (JSONException e) {
418         throw new RuntimeException(e);
419     }
420 }
421
422 },
423 new Response.ErrorListener() {
424     no usages
425     @Override
426     public void onErrorResponse(VolleyError error) {
427         // Gestisci eventuali errori
428         Log.e( tag: "VolleyError", msg: "Errore: richiesta " + error.getMessage());
429     }
430 });
431
432
433 requestQueue.add(stringRequest);

```

## Funzionamento

### 1. Imposta un aggiornamento ogni 5 secondi

- ```
handler.postDelayed(() -> updateScreen(c), 5000);
```

• La funzione viene richiamata ogni 5 secondi per controllare eventuali aggiornamenti della partita.

### 2. Costruisce l'URL della richiesta POST

```

String url1 = url + "getAnyUpdate";
Uri.Builder builder = Uri.parse(url1).buildUpon();
builder.appendQueryParameter("idpartita", idPartita);
String urlWithParams = builder.build().toString();

```

- Crea l'URL per richiedere gli aggiornamenti della partita.
- Aggiunge il parametro **idpartita** con l'ID della partita.
- Il risultato sarà un URL simile a:

<https://tuo-server.com/getAnyUpdate?idpartita=12345>

---

### 3. Effettua una richiesta POST con Volley

- ```
requestQueue = Volley.newRequestQueue(c);
```
- Inizializza una nuova coda di richieste.

```
StringRequest stringRequest = new StringRequest(
    Request.Method.POST, urlWithParams,
    new Response.Listener<String>() { ... },
    new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.e("VolleyError", "Errore: richiesta " + error.getMessage());
        }
    }
);
```

Effettua una richiesta HTTP POST per ottenere gli aggiornamenti dal server.

- Se la richiesta fallisce, viene loggato un errore.
- 

### 4. Gestisce la risposta dal server

- ```
if (response1 == null) {
    Log.d("Carte3", "null...");
}
```
- Se la risposta è null, il gioco continua senza aggiornamenti.

```
JSONObject response = new JSONObject(response1);
String getIdPlayer = response.getString("idFirstPlayer");
```
- Se la risposta contiene dati validi, viene convertita in un oggetto JSON.
- idFirstPlayer indica quale giocatore ha fatto la prima mossa.

- ```

if(getIdPlayer.contentEquals(idPlayer)){
    carta = response.getInt("cartaP2");
    posizione = response.getInt("posCarta2");
} else {
    carta = response.getInt("cartaP1");
    posizione = response.getInt("posCarta1");
}

```

- Determina la carta giocata e la sua posizione, in base a chi ha fatto la mossa.
- 

#### 5. Se l'avversario ha giocato una carta, aggiorna l'interfaccia

- ```

if(carta != -1) { ... }

```
- Se carta è diversa da -1, significa che l'avversario ha giocato una carta.

```

if (posizione == 0) CartaPlayer1 = Carta1Top;
else if (posizione == 1) CartaPlayer1 = Carta2Top;
else CartaPlayer1 = Carta3Top;

if (Nostraposizione == 0) CartaPlaye2 = Carta1Down;
else if (Nostraposizione == 1) CartaPlaye2 = Carta2Down;
else CartaPlaye2 = Carta3Down;

```

- Determina la carta dell'avversario e la posizione della carta scelta dal giocatore.

```

cartaAvversario = carta;
AvversarioPickCarta = true;

```

- Memorizza la carta giocata dall'avversario.

---

#### 6. Se il giocatore non ha ancora giocato, abilita le carte

```

    if (!IoPickCarta) {
        Log.d("alinity","carta diversa da -1 e devo mettere la carta" + IoPickCarta + AvversarioP;
        enableCarts();
        playTurn(c);
    }

```

- Se il giocatore non ha ancora scelto una carta, gli viene permesso di farlo.
- 

## 7. Se entrambi i giocatori hanno scelto, calcola il vincitore

```

else {
    result = compareA(hand,nostraCarta,cartaAvversario);
}

```

- Confronta le carte per determinare il vincitore.

```

if (result == 1) {
    goingTop(CartaPlayer1);
    goingTop(CartaPlaye2);
    punteggioB += hand.getValue(nostraCarta) + hand.getValue(cartaAvversario);
    testoSopra.setText(String.valueOf(hand.getValue(nostraCarta) + hand.getValue(cartaAvversario)));
    CoinsAdd(CoinUp, 1);
    updateCoinValue(coinUptext, punteggioB);
}

```

- Se il giocatore vince il turno:  
Le carte vengono spostate in alto (goingTop()).  
Il punteggio viene aggiornato.

```

else {
    goingDown(CartaPlayer1);
    goingDown(CartaPlaye2);
    punteggioA += hand.getValue(nostraCarta) + hand.getValue(cartaAvversario);
    testoGiu.setText(String.valueOf(hand.getValue(nostraCarta) + hand.getValue(cartaAvversario)));
    CoinsAdd(CoinDown, 1);
    updateCoinValue(coinDownText, punteggioA);
}

```

- Se l'avversario vince, le carte vengono spostate in basso (goingDown()).

```

setCarta(-1, 0);
IoPickCarta = false;
enableCarts();

```

- Resetta lo stato del gioco per il turno successivo.
- 

## 8. Mostra la carta dell'avversario sulla schermata

```
    if (posizione == 0) setClickable(Carta1Top, width/4, height/3, true, carta, false);
    else if (posizione == 1) setClickable(Carta2Top, 0, height/3, true, carta, false);
    else setClickable(Carta3Top, -width/3, height/3, true, carta, false);
```

- Posiziona la carta dell'avversario nella UI.
- 

### Riassunto breve

1. Ogni 5 secondi, chiede al server se ci sono aggiornamenti sulla partita.
  2. Se l'avversario ha giocato, mostra la sua carta.
  3. Se il giocatore non ha ancora giocato, abilita la scelta delle carte.
  4. Se entrambi hanno giocato, calcola il vincitore e aggiorna il punteggio.
  5. Resetta il turno per continuare il gioco.
- 

## Librerie e strutture aggiuntive per lo scambio tra client e API

Per poter effettuare lo scambio JSON richiede che sia server sia client abbiano lo stesso oggetto , quindi sono stati creati seguenti oggetti

- Partita
- Giocatori

### + Libreria esterna Volley

---

In **Android Studio**, puoi gestire i dati in formato **JSON** utilizzando diverse librerie, tra cui **org.json**, **Gson**, e **Volley**. Di default non supporta JSON ed ha bisogno di librerie aggiuntive. Volley è una libreria di **Android** per la gestione delle richieste di rete, sviluppata da Google. È particolarmente utile per gestire chiamate HTTP in modo efficiente, con caching, gestione delle code e risposta rapida.

#### Installazione di Volley

Aggiungi la dipendenza nel file **build.gradle (Module: app)**:

```
dependencies {
    implementation 'com.android.volley:volley:1.2.1'
}
```

Esempio di richiesta GET

```

RequestQueue queue = Volley.newRequestQueue(this);
String url = "https://jsonplaceholder.typicode.com/todos/1";

StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        response -> {
            // Risposta ricevuta con successo
            Log.d("VolleyResponse", response);
        },
        error -> {
            // Errore nella richiesta
            Log.e("VolleyError", error.toString());
        });
queue.add(stringRequest);

```

Esempio di richiesta POST

```

String url = "https://jsonplaceholder.typicode.com/posts";

StringRequest postRequest = new StringRequest(Request.Method.POST, url,
        response -> Log.d("VolleyResponse", response),
        error -> Log.e("VolleyError", error.toString())
) {
    @Override
    protected Map<String, String> getParams() {
        Map<String, String> params = new HashMap<>();
        params.put("title", "Briscola App");
        params.put("body", "Descrizione del progetto");
        params.put("userId", "1");
        return params;
    }
};

queue.add(postRequest);

```

### Vantaggi di Volley

- Semplice da usare
- Cache automatica delle risposte
- Gestione avanzata delle richieste parallele
- Supporta richieste GET, POST, PUT, DELETE

```
5  public class Partita {  
6  
7      4 usages  
8      private String idFirstPlayer;  
9      4 usages  
10     private String idSecondPlayer;  
11     4 usages  
12     private String idPartita;  
13     4 usages  
14     private List<Integer> carte;  
15     3 usages  
16     private String turn;  
17     2 usages  
18     private int cartaP1;  
19     2 usages  
20     private int cartaP2;  
21     2 usages  
22     private boolean partitaFinita;  
23     2 usages  
24     private int cartaP11;  
25     2 usages  
26     private int cartaP12;  
27     2 usages  
28     private int cartaP13;  
29     2 usages  
30     private int cartaP21;  
31     2 usages  
32     private int cartaP22;  
33     2 usages  
34     private int cartaP23;
```

```
1 usages
private int cartaP23;
2 usages
private int posCarta1;
2 usages
private int posCarta2;

2 usages
public Partita() {
}

no usages
public Partita(String idFirstPlayer, String idSecondPlayer, String idPartita, String turn, List<Integer> carte) {
    this.idFirstPlayer = idFirstPlayer;
    this.idSecondPlayer = idSecondPlayer;
    this.idPartita = idPartita;
    this.turn = turn;
    this.carte = carte;
}

no usages
public Partita(String idFirstPlayer, String idSecondPlayer, String idPartita, List<Integer> carte) {
    this.idFirstPlayer = idFirstPlayer;
    this.idSecondPlayer = idSecondPlayer;
    this.idPartita = idPartita;
    this.carte = carte;
}

4 usages
public String getIdFirstPlayer() { return idFirstPlayer; }
```

## Giocatori.class

```
3 public class Giocatori {
4
5     4 usages
6     private String id;
7     2 usages
8     private boolean readyForGame;
9     3 usages
10    private String info;
11
12    no usages
13    public Giocatori(String id, String info) {
14        this.id = id;
15        this.info = info;
16    }
17
18    no usages
19    public boolean isReadyForGame() { return readyForGame; }
20
21    no usages
22    public void setReadyForGame(boolean readyForGame) { this.readyForGame = readyForGame; }
23
24    no usages
25    public Giocatori(String id) { this.id = id; }
26
27    no usages
28    public String getInfo() { return info; }
29
30    no usages
31    public void setInfo(String info) { this.info = info; }
32
33
```

## API

**API che si occupa di ricevere, elaborare e rispondere alle chiamate da parte della App**

**Il controller gestisce le operazioni di gioco della Briscola, tra cui creazione e aggiornamento delle partite, gestione dei giocatori e distribuzione delle carte.**

**Per creare API è stato utilizzato il framework Spring Boot**

**Spring Boot è un framework basato su Spring che semplifica lo sviluppo di applicazioni Java, fornendo configurazioni automatiche e un server integrato.**

**✓ Obiettivo: Eliminare la configurazione manuale di Spring e ridurre la probabilità di errore**

**✓ Ideale per: Microservizi, API REST, applicazioni standalone.**

---

### ◆ Vantaggi di Spring Boot

1. **Configurazione Automatica** → Usa **Spring Boot Starters** per eliminare la configurazione manuale.
2. **Embedded Server** → Include Tomcat, Jetty o Undertow, evitando di installare server esterni.
3. **Dipendenze Gestite** → Usa **Maven** o **Gradle** per gestire le librerie.
4. **Microservizi Ready** → Perfetto per architetture a microservizi.
5. **Monitoraggio & Sicurezza** → Include **Spring Actuator** per il monitoraggio e supporta **Spring Security**.

---

### Funzionalità principali:

1. **Ricerca partita (/getPartita)**
  - Riceve un idPartita via JSON e restituisce la partita corrispondente.
  - Se non esiste, crea una partita di default.
2. **Aggiornamenti partita (/getAnyUpdate)**
  - Riceve un idPartita e restituisce la partita aggiornata.

- 3. Inserimento dati (/putData)**
    - Aggiorna una partita esistente nella lista listaPartite.
  - 4. Aggiornamento mosse (/updateData)**
    - Registra la carta giocata da un giocatore, modificando cartaP1 o cartaP2.
  - 5. Gestione giocatori e partite (/ready)**
    - Se un giocatore è già in attesa, restituisce "wait".
    - Se ci sono due giocatori disponibili, crea una nuova partita.
- 

#### Metodi interni

- **readyForGame():** Assegna due giocatori a una nuova partita, distribuisce carte e imposta il turno iniziale.
- **carte():** Genera e mescola le 40 carte del mazzo.

#### Struttura dati

- **listaPartite:** Contiene le partite attive.
- **listaplayer:** Contiene i giocatori in attesa.
- **pgmap:** Mappa tra ID giocatori e ID partita.
- **StringListaPlayer:** Contiene lista giocatori in attesa di una partita
- **Giocatori player1, player2**

Questo controller implementa le API per gestire il matchmaking, la creazione delle partite e l'aggiornamento delle mosse nel gioco della Briscola.

```

12  @Controller no usages
13  @RequestMapping("/post")
14  public class PostApiController {
15      private static List<Giocatori> listaPlayer = new ArrayList<>(); 5 usages
16      private static List<Partita> listaPartite = new ArrayList<>(); 7 usages
17      private Map<String, String> pgmap = new HashMap<>(); 4 usages
18      private static List<String> Stringlistaplayer = new ArrayList<>(); 2 usages
19      private Giocatori player1, player2; 6 usages
20
21      @PostMapping("/getPartita") no usages
22      @ResponseBody
23      public Partita findPartitaById(
24          @RequestBody JsonPartita partitajson) {
25          List<Integer> l = carte();
26          Partita p1 = new Partita( idFirstPlayer: "id1", idSecondPlayer: "id2", idPartita: "id1id2",l);
27          for (Partita partita : listaPartite){
28              if (partita.getIdPartita().contains(partitajson.getIdPartita())){
29                  return partita;
30              }
31          }
32          return p1;
33      }
34
35      @PostMapping("/getAnyUpdate") no usages
36      @ResponseBody
37      public Partita getAnyUpdate(
38          @RequestParam String idpartita) {
39          for (Partita partita : listaPartite){

```

```

38      @RequestParam String idpartita) {
39          for (Partita partita : listaPartite){
40              if (partita.getIdPartita().contains(idpartita)){
41                  return partita;
42              }
43          }
44          return null;
45      }
46
47      @PostMapping("/putData") no usages
48      @ResponseBody
49  @ public String postResponseController2(
50      @RequestBody Partita p) {
51      System.out.println("arrived data from "+p.getTurn()+"\n"+p.toString());
52
53      for (Partita partita : listaPartite){
54          if (partita.getIdPartita().contains(p.getIdPartita())){
55              listaPartite.remove(partita);
56              listaPartite.add(p);
57          }
58      }
59      return "inserted";
60  }
61      @PostMapping(value = "/updateData") no usages
62      @ResponseBody
63      public Partita updateData(@RequestParam JsonPartita partitaJson){
64
65          for (Partita partita : listaPartite){
66              if (partita.getIdPartita().contains(partitaJson.getIdPartita())){

```

```
66     if (partita.getIdPartita().contains(partitaJson.getIdPartita())){
67         if(partita.getIdFirstPlayer().contains(partitaJson.getGiocatore())){
68             partita.setCartaP1(partitaJson.getCarta());
69             partita.setPosCarta1(partitaJson.getPos());
70         }
71         else {
72             partita.setCartaP2(partitaJson.getCarta());
73             partita.setPosCarta2(partitaJson.getPos());
74         }
75         return partita;
76     }
77 }
78
79     return null;
80 }
81
82
83 @PostMapping(value = "/ready")  no usages
84 @ResponseBody
85 public String rForGame(@RequestParam String idp){
86     System.out.println(idp);
87     //    for (Partita partita : listaPartite){
88     //        if (partita.getIdPartita().contains(idp)){
89     //            return idp;
90     //        }
91     //    }
92     if(pgmap.containsKey(idp)) return pgmap.get(idp);
93     if(Stringlistaplayer.contains(idp)) return "wait";

```

```

85     public String rForGame(@RequestParam String idp){
91     // }
92     if(pgmap.containsKey(idp)) return pgmap.get(idp);
93     if(Stringlistaplayer.contains(idp)) return "wait";
94     else{
95         listaplayer.add(new Giocatori((idp)));
96         Stringlistaplayer.add(idp);
97     }
98
99     if(listaplayer.size()%2 == 0){
100        String idPartita = readyForGame();
101        System.out.println("after ready"+idPartita);
102        return idPartita;
103    }
104    return "wait";
105
106}
107// @PostMapping("/newPlayer")
108// @ResponseBody
109// public String postResponseController3(
110//     @RequestBody String id) {
111//
112//
113//     listaplayer
114//     List<Integer> l = carte();
115//     Partita p1 = new Partita("id1","id2","id1id2",l);
116//     return p1;
117// }
118
119
120    private String readyForGame(){ 1 usage
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146 @
147     private List carte(){ 2 usages
148         List<Integer> carte = new ArrayList<>();

```

```
146 @    private List carte(){ 2 usages
147         List<Integer> carte = new ArrayList<>();
148         for (int i = 0; i < 40; i++) {
149             carte.add(i);
150         }
151         // Mescola la lista in ordine casuale
152         Collections.shuffle(carte);
153         return carte;
154     }
155 }
```

## Test e Risultati

L'applicazione è stata sottoposta a diversi test per garantire la qualità del prodotto finale:

- **Test Funzionali:** Verifica del corretto funzionamento delle regole di gioco e delle interazioni utente.
- **Test di Usabilità:** Valutazione dell'interfaccia con un campione di utenti, raccogliendo feedback per miglioramenti.
- **Test di Prestazioni:** Ottimizzazione delle animazioni e del caricamento delle risorse per garantire fluidità anche su dispositivi meno performanti.

I risultati dei test hanno evidenziato un alto grado di soddisfazione degli utenti, con particolare apprezzamento per l'intuitività dell'interfaccia e la fedeltà alle regole tradizionali della Briscola.

## Conclusioni e Sviluppi Futuri

Il progetto ha dimostrato come sia possibile adattare un gioco tradizionale come la Briscola a una piattaforma digitale, offrendo un'esperienza moderna senza perdere l'essenza originale. Tra i possibili sviluppi futuri si segnalano:

- **Personalizzazione:** Possibilità di scegliere mazzi di carte con design regionali o tematici.
- **Estensione a Piattaforme iOS:** Creazione di una versione dell'app per dispositivi Apple.

Con questo lavoro, si auspica di contribuire alla valorizzazione del patrimonio ludico italiano, rendendolo accessibile alle nuove generazioni attraverso la tecnologia.

---

## Codice sorgente



---

## Bibliografia

- Documentazione ufficiale di Android Studio e Java.
  - Linee guida di Google Material Design.
  - Risorse online sulla storia e le regole del gioco della Briscola.
- 

## Terminologia

- **API** - application programming interface
- **Framework** - raccolta di componenti software riutilizzabili che rendono più efficiente lo sviluppo di nuove applicazioni.
- **IDE** - integrated development environment
- **JSON** - a standard text-based format for representing structured data based on JavaScript object syntax