# Relazione Applicazioni Mobili – Sette e mezzo

Componenti del gruppo:

- Ricciulli Pasquale 0000923980 pasquale.ricciulli@studio.unibo.it (INF)
- Lucarelli Enrico 0000922471 enrico.lucarelli4@studio.unibo.it (INFMAN)

Repository:

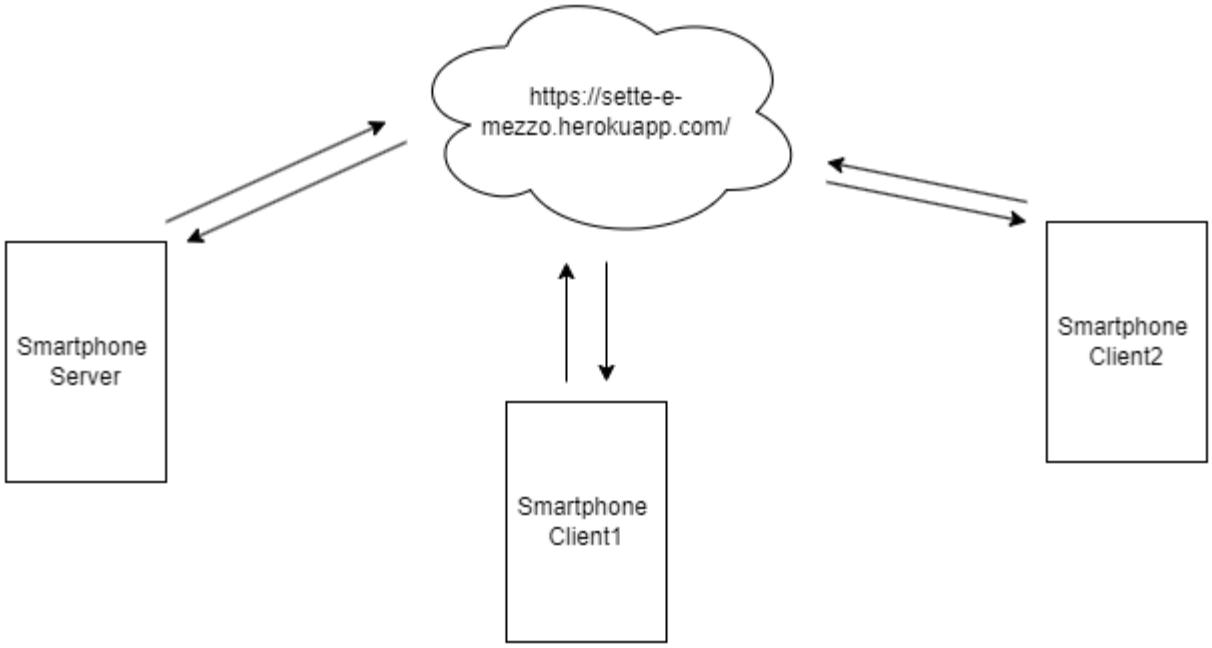
Il gioco che abbiamo realizzato è una versione training del famoso gioco di carte Sette e mezzo. All'inizio della partita ogni giocatore riceve una carta che non viene mostrata agli avversari. I giocatori, quando sono di turno, possono scegliere se riceve un'altra carta o fermarsi. Vince chi ha raggiunto il valore più vicino a 7.5, se si supera il valore si perde. Le carte da 1 a 7 assumono il valore che rappresentano, le carte da 8 a 10 valgono 0.5. In caso di parità tra il mazziere e un giocatore vince il mazziere.

## Client/Server

La prima scelta effettuata è stata su quali compiti attribuire al server e se client e server dovessero essere due app distinte. Come anche indicato nelle specifiche abbiamo affidato all'app server la gestione del mazzo, è infatti suo compito estrarre le carte ed inviarle ai client. Per quanto riguarda la scelta del numero di app da implementare abbiamo scelto di svilupparne una sola e dare la possibilità all'utente di scegliere in quale ruolo giocare di volta in volta. Si può infatti decidere di giocare nel ruolo di mazziere (app server) o di giocatore (app client).

## Comunicazione

Per la comunicazione tra le varie app abbiamo utilizzato la seguente struttura:




All'apertura dell'app essa si collega ad un'applicazione web che funge da passa carte tra le app. Abbiamo fatto questa scelta per poter rendere più realistico il gioco e per poterlo utilizzare anche al di fuori della stessa rete locale (LAN).

## Server Node e Socket.io

Node.js è un framework che viene usato per scrivere applicazioni in javascript lato client.

Come primo passo abbiamo inizializzato la porta di ascolto, l'ip del server e creato una classe game per salvare i dati delle partite.

Per la gestione delle connessioni e lo scambio di dati tra client e server abbiamo utilizzato socket.io, una libreria di rete Javascript e Node.js eseguita lato server che consente una comunicazione in tempo reale da client a server e viceversa.

Queste sono le funzioni create per gestire lo scambio di dati.

Ad esempio, la funzione deletePlayer serve ad eliminare un player dal game quando alla fine di una partita viene scelta l'opzione non giocare ancora.

```
socket.on('confGame',confGame);
socket.on('createGame',createGame);
socket.on('joinGame',joinGame);
socket.on('startGame',startGame);
socket.on('sendFirstCard',sendFirstCard);
socket.on('giveMeCard',giveMeCard);
socket.on('terminateTurn',terminateTurn);
socket.on('sendCard',sendCard);
socket.on('closeRound',closeRound);
socket.on('isYourTurn',isYourTurn);
socket.on('overSize',overSize);
socket.on('continueGame',continueGame);
socket.on('deletePlayer',deletePlayer);
socket.on('deleteGame',deleteGame);
socket.on('saveWinner',saveWinner);
```

```
public class SocketClass {
    private static Socket socket = null;
    public void connection(){
        try {
            socket = IO.socket( uri: "https://sette-e-mezzo.herokuapp.com");
            socket.connect();
        }catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }

    public boolean isConnected(){ return socket.connected();}
    public void disconnection() { socket.disconnect(); }
    public Socket getSocket() {
        if(socket == null)
        {
            connection();
        }
        return socket;
    }
    public String getId(){ return socket.id();}
}
```

Per quanto riguarda la gestione della connessione nei dispositivi mobili, utilizziamo socket.io e per la gestione del socket abbiamo realizzato la classe SocketClass.java.

Il metodo Connection che si occupa di creare la connessione con il server node.

Il metodo isConnected permette di controllare se il client è già connesso così da non dover creare un nuovo socket.

Il metodo disconnected server per chiudere la connessione.

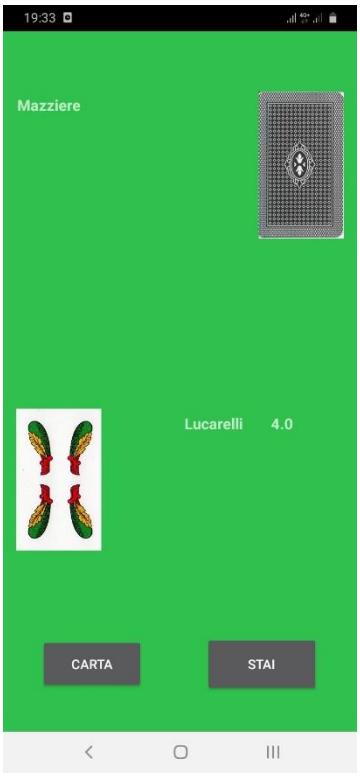
Il metodo getId serve per prendere l'id socket associato al client.

## Server node e avvio della partita

Il compito del server node, oltre a quello, già anticipato, di passa carte tra le app è gestire la creazione e il popolamento delle stanze di gioco. L'app server sceglie che nome dare alla partita, il numero massimo di giocatori che vorrebbe in essa ed invia al server node queste informazioni che la crea sulla base di quei dati. I client inviano invece l'username dell'utente e il numero di giocatori che dovrebbe avere la stanza. Compito del server node è aggiungere i client ad una stanza dal numero di giocatori scelto, se disponibile. Abbiamo scelto che la partita venga avviata esclusivamente dal server in modo da potergli dare anche la possibilità di avviarla con un numero di giocatori minore.

## Room

Il mazziere alla creazione della partita può scegliere un numero di giocatori tra 2 a 4. A seconda dei giocatori che saranno collegati alla partita si apriranno Activity differenti con layout differenti.





## Elementi utilizzati

Il mondo android fornisce numerosi elementi, nel corso dello sviluppo della nostra app ne abbiamo utilizzati solo una piccola parte. Tra questi vi sono:

- Activity: elemento di base di ogni applicazione, lo abbiamo utilizzato per realizzare le varie schermate
- RecyclerView: le abbiamo utilizzate per visualizzare le carte aggiuntive richieste dal giocatore. Di conseguenza abbiamo utilizziamo anche gli Adapter
- Dialog: abbiamo realizzato un Dialog personalizzato per permettere ai client di visualizzare l'esito della partita e per chiedere loro se vogliono continuare a giocare
- Toast: gli abbiamo utilizzati per indicare agli utenti comportamenti non consentiti (es. Avvio partita senza nome)
- Notification: abbiamo realizzato una notifica che viene mostrata due giorni dopo l'accesso all'app in cui si invita l'utente a tornare a giocare

## Strumenti di lavoro

Nello sviluppo dell'app oltre all'ide Android Studio abbiamo utilizzato altri strumenti tra questi:

- L'IDE Visual Studio per editare e startare il server node prima che avvenisse il deploy
- Repository GitHub per il versioning del codice sorgente
- Draw.io per la realizzazione di schemi di connessione

## Suddivisione lavoro

Durante lo sviluppo dell'app abbiamo cercato di suddividerci il lavoro in maniera equa. Le suddivisioni riportate di seguito sono ovviamente indicative, ci siamo confrontati su ogni componente dell'app, sia per definire le scelte implementative sia per l'eventuale correzione di bug.

Ricciulli: si occupato dello sviluppo iniziale del server node stabilendo al connessione e creando i primi metodi per costruire le stanze di gioco; ha implementato la classe SocketClass per la gestione dell'istanza del connessione; ha sviluppo la versione del gioco a 3 player realizzando sia l'Activity lato Client che quella lato Server con i relativi layout; ha realizzato le Activity che intercorrono dall'apertura dell'app all'inizio della partita (MenuActivity, SetGameActivity, WaitActivity, WaitRoomActivity, PlayerActivity); ha implementato la logica ed il meccanismo per il restart delle partite;

Lucarelli: ha sviluppo la versione del gioco a 2 e 4 player realizzando sia leActivity lato Client che quelle lato Server con i relativi layout; ha implementato la notifica e il BroadcastReceiver che la contiene; ha realizzato il dialog personalizzato in cui viene visualizzato l'esito e viene chiesto al client se vuole continuare a giocare; ha realizzato la parte Model implementando la classe Deck e quella Card; ha realizzato la classe Utils contenente metodi risultati utili nel corso dello svilippo e le stringhe per limitare l'hardcoding.

## Deploy server node

Al fine di rendere utilizzabile l'app anche al di fuori di una stessa rete locale abbiamo caricato il server node su una piattaforma di servizi cloud, Heroku. Essa ci ha fornito un indirizzo web a cui collegarci per poter utilizzare il server node. Inserito nelle app è possibile giocare ovunque. Su Heroku è possibile collegarsi ad una repository GitHub e fare il deploy, per questo motivo abbiamo creare una nuova repository con all'interno solo il server node.

https://github.com/PasqualeRic/Sette-e-mezzo-server

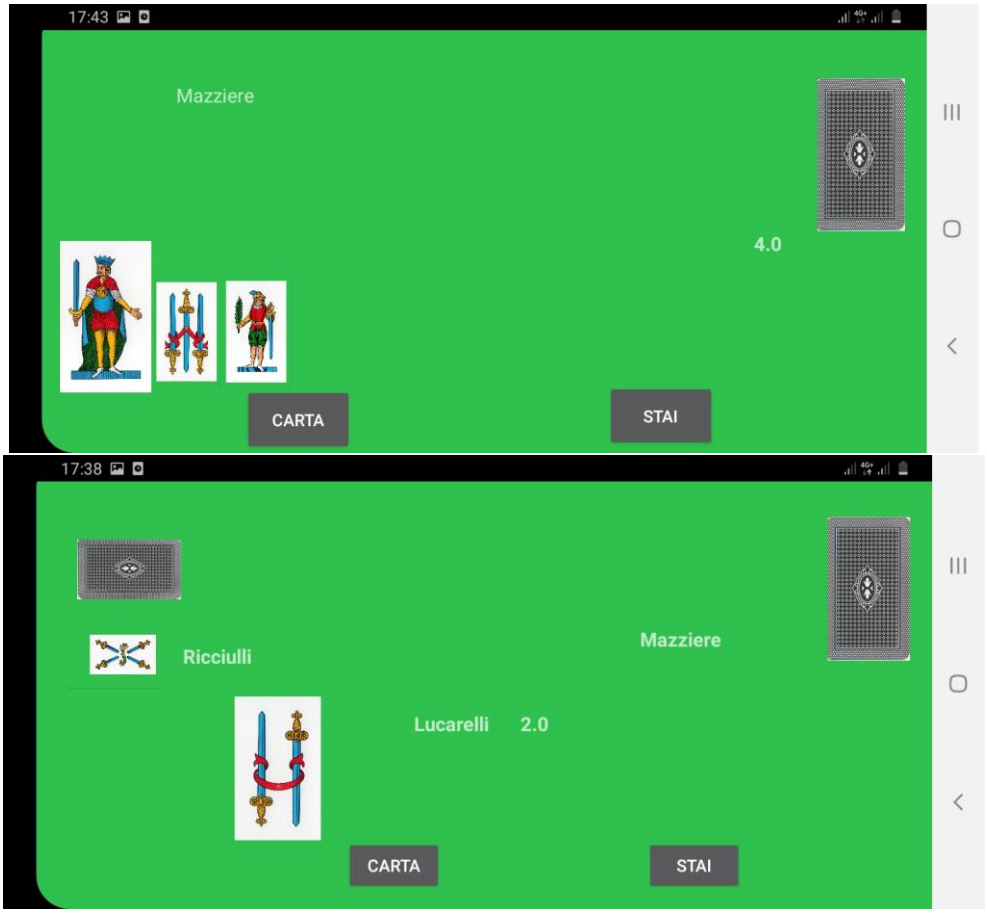
## Test su dispositivi reali

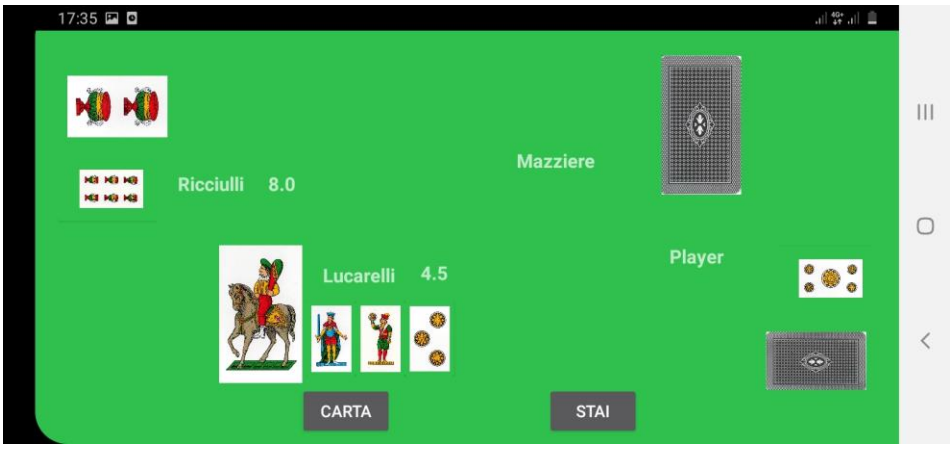
Nel corso dello sviluppo dell'app oltre al debug svolto con gli emulatori, abbiamo anche avuto l'occasione di effettuare dei test su dei dispostivi reali.

- Samsung Galaxy A30s
- Samsung Galaxy S20
- Samsung Galaxy A12
- Redmi Note 7
- Samsung Galaxy M12

## Landscape

L'applicazione è stata progettata per essere eseguita anche il landscape, di conseguenza ogni activity ha due layout. Ecco come si mostrano le activity di gioco nei vari casi.






Durante la rotazione l'activity viene eliminata e ricreata, per questo per poter consentire all'utente di continuare la partita in corso abbiamo dovuto fare l'override dei metodi `onSaveInstanceState` e `onRestoreInstanceState`.

```java
@Override
protected void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);

    // Player
    outState.putString(Utils.idFirstCard,idFirstCard);
    outState.putStringArrayList("myCards", Utils.getIdCards(myCards));
    outState.putDouble("myScore",myScore);

    //Dealer
    outState.putString("idFirstCardDealer",idFirstCardDealer);
    outState.putStringArrayList("dealerCards", Utils.getIdCards(dealerCards));
    if(scoreDealer!=null)
        outState.putDouble("scoreDealer",scoreDealer);

    outState.putString("tvResult",tvResult.getText().toString());

    outState.putBoolean("isMyTurn",isMyTurn);

}
```

```java
@Override
protected void onRestoreInstanceState(@NonNull Bundle savedIstanceState){
    super.onRestoreInstanceState(savedIstanceState);

    // Player
    idFirstCard = savedIstanceState.getString(Utils.idFirstCard);
    myCards = Utils.getCardsById(savedIstanceState.getStringArrayList( key: "myCards"));
    myScore = savedIstanceState.getDouble( key: "myScore");

    ivMyFirstCard.setImageResource(Deck.getIstance().getCardById(idFirstCard).getIdImage());
    tvMyScore.setText(""+myScore);

    myCardAdapter = new CardAdapter(myCards);
    myRecyclerView.setAdapter(myCardAdapter);

    tvResult.setText(savedIstanceState.getString( key: "tvResult"));

    isMyTurn = savedIstanceState.getBoolean( key: "isMyTurn");
    if(!isMyTurn){
        btnCarta.setVisibility(View.INVISIBLE);
        btnStai.setVisibility(View.INVISIBLE);
    }
    // Dealer
    idFirstCardDealer = savedIstanceState.getString( key: "idFirstCardDealer");
    dealerCards = Utils.getCardsById(savedIstanceState.getStringArrayList( key: "dealerCards"));

    cardAdapterDealer = new CardAdapter(dealerCards);
    dealerReyclerView.setAdapter(cardAdapterDealer);
```

## Model

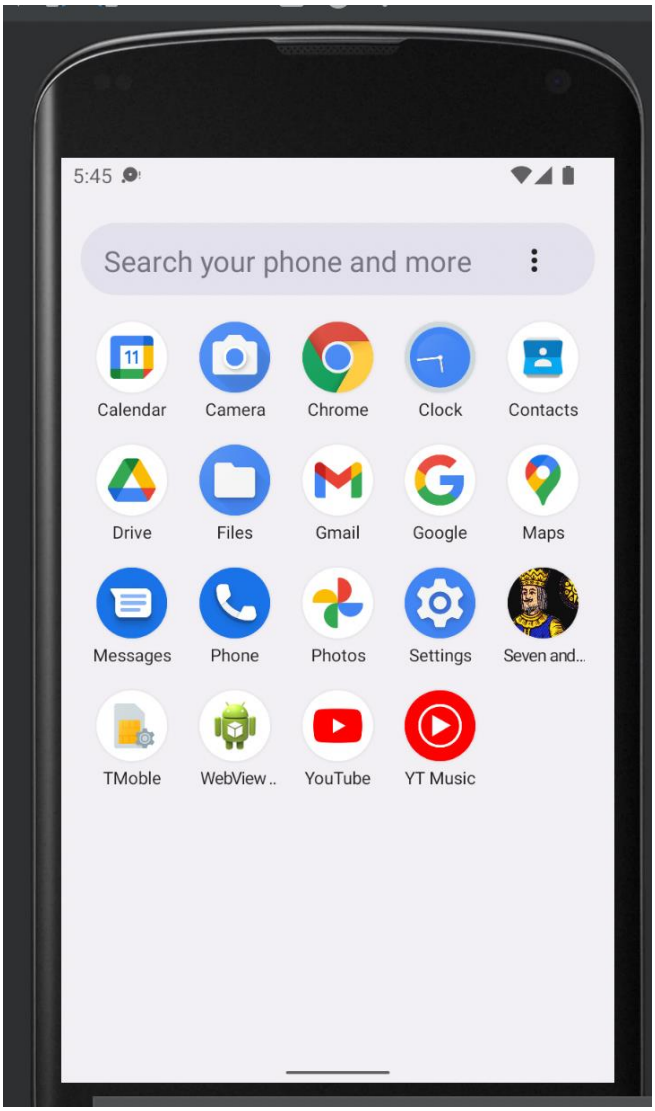
Per quanto riguarda il mazzo di carte e la carta in sé abbiamo creato due classi, la classe Card che è l'istanza di una singola carta dove salviamo l'id, immagine e valore.

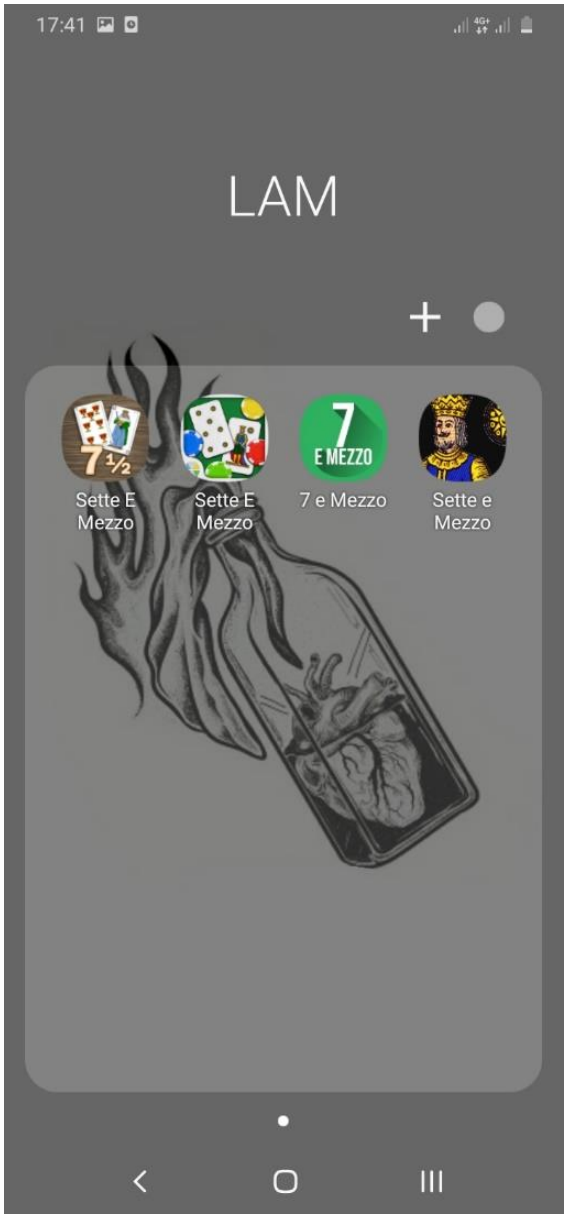
```java
public class Card{
    private String id;
    private int idImage;
    private double value;
    private boolean extracted;

    public Card(String id,int idImage,double value) {
        extracted = false;
        this.id=id;
        this.idImage=idImage;
        this.value=value;
    }

    // GETTER
    public String getId(){return id;}
    public double getValue(){return value;}
    public boolean isExtracted(){return extracted;}
    public int getIdImage(){return idImage;}

    // SETTER
    public void setExtracted(){ extracted=true;}

    public JSONObject toJSON(){
        JSONObject json = new JSONObject();
        try {
            json.put(Utils.id, id);
            json.put(Utils.value, value);
            json.put( name: "idImage", idImage);
        }catch(Exception e){}
        return json;
    }
}
```

La classe Deck rappresenta il mazzo e utilizza il pattern singleton che ci garantisce che ci sia sempre un'unica istanza di questa classe, questo ci server per tenere traccia delle carte che sono già uscite così da non ripescare le stesse carte durante la partita.

## Icona

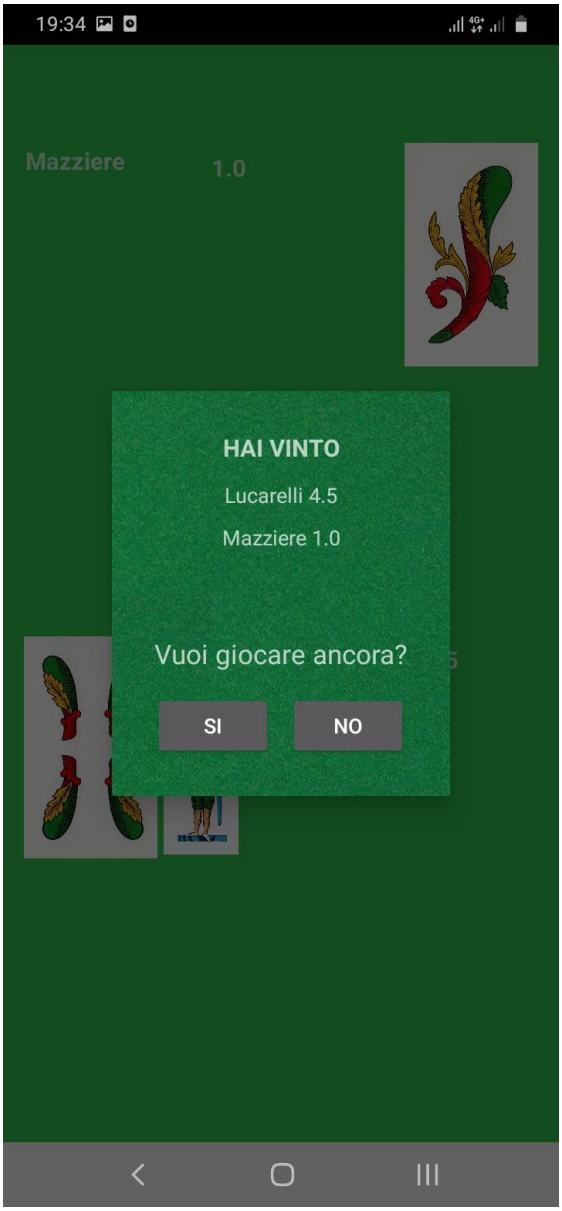
Durante lo sviluppo dell'app abbiamo deciso di personalizzare l'icona, per fare ciò abbiamo modificato all'interno del manifest , icon e Round icon con un'immagine a nostra scelta.



## Visualizzazione esito

Come già anticipato al termine di una mano di gioco i risultati vengono mostrati in un dialog personalizzato. Ecco il risultato:




## Test su diverse versioni di Android

Nel corso dello sviluppo dell'app abbiamo cercato di testare l'app su diverse versioni del sistema operativo android. Abbiamo usato dalla versione con API 33 fino a quella con API 23.