Bingo Party

Laboratorio di Applicazioni Mobili A.A. 2021/22

Filippo Brajucha – 920975
filippo.brajucha@studio.unibo.it
https://github.com/benzebra/BingoParty

Ottobre 2022

Indice

1 Introduzione

Il progetto che viene illustrato ha lo scopo di sviluppare il gioco del Bingo (o della Tombola) in versione *peer-to-peer* per smartphone Android.
Il numero minimo di giocatori è 2 e il numero minimo di dispositvi necessari è 3, infatti ci saranno i 2 giocatori (identificati come client) e il server.
Il **server** si occupa di aprire la connessione e gestire le dinamiche di gioco (estrazione numeri, distribuzione cartelle di gioco e controllo della vincita).
Il **client**, una volta collegato al server, non fa altro che ricevere i suoi ordini sottoforma di messaggi e li utilizza per renderizzare oggetti grafici o gestire le dinamiche di gioco.
Le dinamiche di gioco sono molto semplici e si ispirano al gioco del Bingo, per certi aspetti differente dalla Tombola (dato che non esiste la possibilità di fare ambo, terno e cinquina). I giocatori inseriscono il proprio username e chiedono di connettersi al server, una volta connessi ricevono le cartelle (da 1 a 4) e si avvia la partita. Le cartelle non sono altro che delle matrici 3x3 contententi numeri interi casuali, diversi l'uno dall'altro e compresi tra 1 e 90. I numeri estratti dal server ogni 2,5 secondi vengono segnati sulle tabelle (ove presenti) e vengono mostrati nella schermata principale di gioco.
Non esiste una modalità single-player ma è solo disposto il multi-player sfruttando il protocollo **TCP/IP**, viene usata l'architettura client - server, implementata attraverso i pacchetto `java.net.Socket` e `java.net.ServerSocket`.
Il progetto è realizzato interamente in Java, nell'ambiente di Android Studio.

2 Architettura

In generale, come da specifica, il progetto può essere suddiviso in tre sezioni: Discovery Phase (fase iniziale in cui il server attende che i client si connettano), Playing Phase (fase di gioco vera e propria) e Conclusion Phase (fase conslusiva con annunciazione del vincitore). Ovviamente ogni singola fase appare in modo differente a seconda dell'applicazione che stiamo utilizzando (client o server).
Qui di seguito vengono esplicate in maniera più completa le tre fasi, suddivise anche tra client e server e arricchite con snippet di codice e screenshot delle varie schermate.

2.1 Server

Il server funge da "cervello" di gioco per i vari client che vi si collegano. Esso si occupa di mettere in comunicazione (e quindi in competizione) i client, genera le cartelle e le distribuisce e genera e distribuisce i numeri estratti. Esso tiene traccia dell'evoluzione di gioco ogni volta che esce un numero.
Un punto centrale della struttura del server è l'utilizzo della classe "Player", una classe creata *ad-hoc* in modo da poter facilitare la comunicazione con tutti i giocatori limitando il più possibile ritardi ed errori. I giocatori di una certa sessione di gioco, infatti, sono parte di un ArrayList di Player.
La classe player contiene 8 elementi di diverso tipo e utili al fine di facilitare la comunicazione client-server.

```
PrintWriter sender;
BufferedReader receiver;
String address;
Receiver threadReceiver;
Socket clientSocket;
String name;
int matrixNumber;
ArrayList<int[]> matrixArray;
```

Figura 1: Classe Player

Ogni elemento della classe prevede i propri *getter* e *setter*, importante da sottolineare l'oggetto `threadReceiver` di tipo `Receiver`, un'altra classe personalizzata che estende la classe `Thread` di Java (in modo da poter essere runnato) e che permette di ricevere i messaggi dal client e tradurli in azioni o comportamenti (a seconda del messaggi stesso) e facilitando la sincronizzazione senza incorrere in Exception particolari o costrutti di sincronizzazione più complessi.
L'ArrayList di Player viene creata nella `LobbyActvity`, mano a mano che i vari giocatori si connettono al server; inizialmente i player creati possiedono solo `address` (indirizzo ip, `InetAddress`) e `clientSocket`, successivamente vengono settati anche `sender` e `receiver` e `threadReceiver`, gli altri dati vengono impostati dopo.

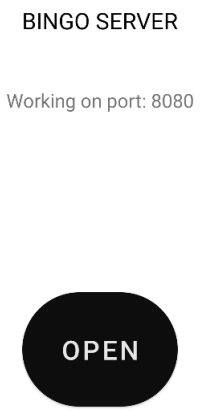


Figura 2: MainActivity

2.1.1 Discovery Phase

Fase di connessione e collegamento, il server si mette a disposizione per le connessioni dei vari client.

MainActivity La MainActivity contiene semplicemente il pulsante "START" per fare partire la connessione (anche se in realtà la ServerSocket si attiva nell'activity successiva).

LobbyActvity Activity principale della Discovery Phase del server. Dalla LobbyActivity, infatti, viene attivato il thread `connection` che avvia la ServerSocket su una certa `PORT` e le imposta un timeout di 10 secondi entro il quale i client devono necessariamente connettersi.
In questa activity viene anche istanziata l'ArrayList di Player, durante ogni singola connessione (fase in cui la ServerSocket accetta le connessioni esterne) il server istanzia un oggetto Player con relativo `sender`, `receiver`, `threadReceiver`, `address` e `clientSocket`, solo successivamente vengono collezionati gli altri dati (come nome, numero di cartelle, lista degli array corrispondenti alle cartelle).
Una volta scaduto il tempo a disposizione per la connessione (quindi il thread finisce in SocketTimeoutException), il server si occupa di ricevere i nomi dei giocatori e aggiungerli al loro oggetto all'interno della lista.

L'ultimo controllo è sul numero di giocatori (ovviamente si arriva a questo controllo solo dopo che il server ha recuperato tutti i loro nomi), se e solo se è maggiore o uguale a due (`playersNumber` $\geq$ `2`),il gioco può partire e viene attivato il tasto "start", altrimenti viene chiusa la connessione. Si esce forzatamente dalla LobbyActivity in 3 modi:

1. Timeout della socket (dopo 10 secondi)
2. Numero massimo di giocatori raggiunto (4)
3. Premuto il pulsante "Close Connection"

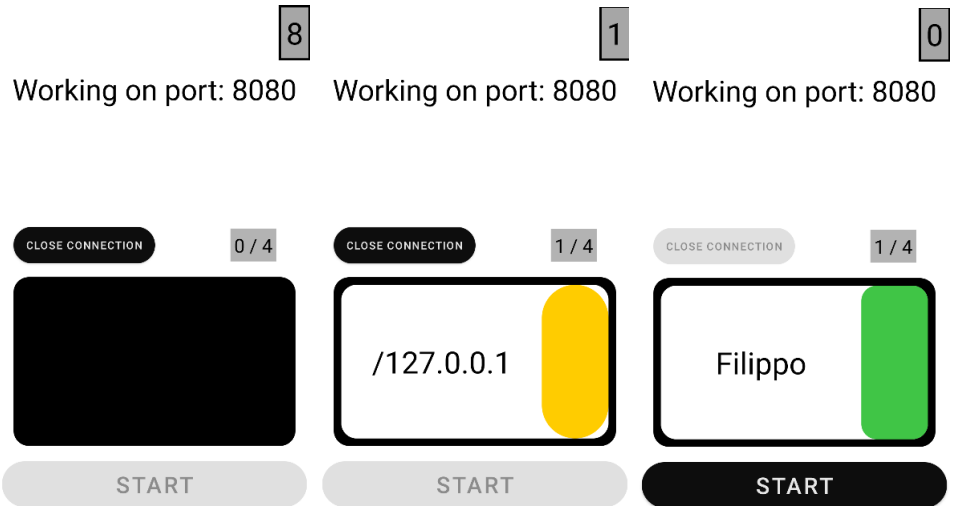
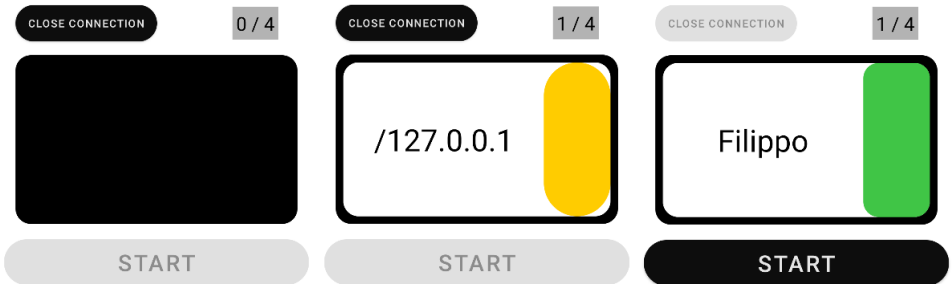


Figura 3: LobbyActvity

RecyclerView e Adapter Nella parte grafica di questa Activity è presente anche una RecyclerView (`R.id.recyclerViewPlayers`), in essa vengono visualizzati i giocatori connessi, viene aggiornata in tempo reale con ogni nuova connessione. Inizialmente viene visualizzato l'`address` con un bollino giallo, una volta inserito il nome e recuperato da parte del server, l'indirizzo viene cambiato con esso e il bollino diventa verde.



PreGameActivity La PreGameActivity si occupa di 3 azioni principali:

1. Creare la stream di numeri da estrarr
2. Recuperare il numero di cartelle per ogni giocatore, dato inserito manualmente da ognuno e inviato al server ($1 <$ `numero cartelle` $<$ 5)
3. Inoltrare le cartelle al client corretto (dopo averle generate casualemente, con gli opportuni controlli come l'assenza di numeri doppi nella stessa cartella)

La prima azione viene eseguita daL metodo `createField()`.
Le altre due vengono eseguite all'interno del thread `sendStartAndMatrix`, come dice il nome, il thread si occupa di inviare il segnale di "start" al client e di inviargli anche le matrici (che rappresentano le cartelle di gioco). Inoltre, grazie a questo ultimo thread, vengono anche inseriti i dati all'interno dell'ArrayList di giocatori creata in precedenza.

2.1.2 Playing Phase

Fase di gioco vera e propria, le varie cartelle sono gia impostate, la stream di numeri casuali che deve uscire è gia stata preparata (ho preferito farlo in precedenza in modo da favoirire la sincronicità tra client e server e diminuire i ritardi dovuti al controllo sui numeri casuali gia usciti).

GameLoopActivity Nella GameLoopActivity avviene la vera e propria dinamica di gioco.
Come prima cosa viene inviato a tutti i client il segnale di avvio del gioco sottoforma di messaggio. Solo successivamente viene attivato il thread

`gameLoop` che contiene tutte le fasi di gioco che vengono eseguite in loop fino a quando non viene trovato un vincitore:

1. Resettare a 0 il numero di giocatori controllati
2. Estrarre il numero
3. Inserire il numero nella grafica (sia quella principale che quella dei numeri estratti di recente)
4. Avviare un thread di controllo di vittoria (`checkWinThread`) per ogni giocatore
5. Aggiornare la grafica con le cartelle di ogni giocatore

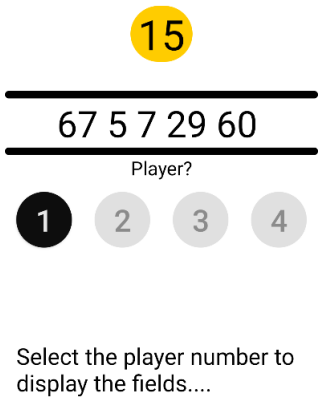


Figura 4: GameLoopActivity

Fragment_Player Nella grafica dell'activity GameLoop sono presenti anche 4 pulsanti che permettono la vista delle cartelle dei giocatori, a seconda del pulsante che clicco, se abilitato, posso vedere le cartelle aggiornate ad ogni estrazione di un numero.
Questa grafica è composta da una RecyclerView inserita all'interno di un fragment. Per creare il fragment, infatti, è necessario dare in input il giocatore selezionato, da esso vengono estratti il **nome** e la lista di **cartelle**.

Entrambi i dati vengono visualizzati a schermo. L'aggiornamento delle cartelle avviene tramite eliminazione del numero, sia per un fatto di semplicità strutturale (visto che l'algoritmo implementato dal checkwin funziona così) che per un discorso grafico (risulta più d'impatto se vedo i numeri sparire man mano che vengono estratti).

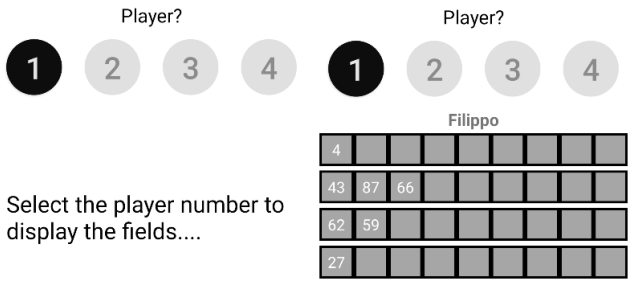


Figura 5: PlayerFragment

checkWinThread Thread che viene richiamato dal GameLoop, prende in input un giocatore (e quindi la sua relativa ArrayList di `int[]` che rappresentano le cartelle) e un dato numero (numero uscito in quel momento e da controllare se rappresenta l'occasione di vinecere per un qualsiasi giocatore). Non fa altro che ciclare su tutta la lunghezza degli array e controlla se il numero è presente; in caso di risposta affermativa elimina quella casella e accorcia l'array (ovviamente di 1 elemento), se la lunghezza diventa 0 quel giocatore ha fatto bingo, altrimenti si procede con l'array successivo.
Ogni thread controlla un solo giocatore e imposta il suo stato su "checked", solo una volta che il numero di giocatori controllati è uguale al numero di giocatori totali ricomincia il Game Loop.

2.1.3 Conslusion Phase

Fase finale in cui viene annunciato il primo tra i giocatori che è riuscito a completare una cartella (e quindi ha fatto "BINGO!").
Il gioco si può concludere anche per timeout (dopo 180000ms) oppure perchè tutti i numeri sono stati estratti ma nessun giocatore ha chiamato il Bingo.

WinnerActivity Questa activity è molto semplice perchè viene richiamata dal Game Loop che gli passa addirittura il nome del vincitore e semplicemente si occupa di mostrare a schermo il vincitore.
Per poter vincere è necessario che il giocatore, oltre ad aver completato almeno una cartella, prema il pulsante "BINGO". Una volta premuto notifica il server che fa partire la WinnerActivity (notificando poi, anche tutti gli altri giocatori).

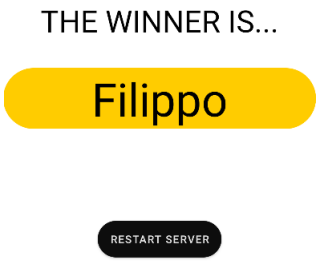


Figura 6: WinnerActivity

2.2 Client

Il client si presenta solo come interfeccia grafica che interpreta i comandi del server arrivati tramite messaggo, non fa altro che istanziare la connessione con il server e illustrare ciò che esso ha creato e deciso.
Il client, infatti, riceve dal server tutte le indicazioni per procedere in avanti e attivare le altre activity, riceve anche lo stream di uscita dei numeri e le cartelle di gioco, oltre all'avviso di vittoria di un certo giocatore.
Gli unici momenti in cui è il client ad inviare un messaggio utile al server è nella Discovery Phase, esso gli invia sia il nome utente che il numero di cartelle richieste.
Una classe di supporto molto utile creata nel client è la classe `Matrix` con la sua sotto-classe `MatrixNumbers`, sono spiegate meglio nel paaragrafo dedicato

2.2.1 Receiver

Per il client è importante menzionare il `Receiver`, un thread che viene attivato fin dall'inizio, non appena avviene la connessione con il server.
Esso riceve i messaggi dal server e li traduce in azioni facendo riferimento a metodi statici all'interno delle activity, ad esempio quando riceve "`start`", provvede a settare un flag nella LobbyActivity per passare all'activity successiva, oppure quando riceve "`matrixstart`" so che il server nel messaggio successivo invierà le cartelle sottoforma di stream di caratteri e quindi predispongo dei metodi per memorizzarle e passarle all'activity che ne necessita.

2.2.2 Discovery Phase

Fase di ricerca in cui il client apre la sua Socket e si mette alla ricerca di un ServerSocket su una certa `PORT`, una volta istanziata la nuova connessione ottiene tutti i dati necessari per poter attivare `sender` e `receiver` così da poter restare in costante comunicazione con il server.

MainActivity La MainActivity è semplicemente la schermata iniziale, ogni giocatore inserisce il proprio nome e, schiacciando il pulsante, avvia la LobbyActivity.

Figura 7: MainActivity

LobbyActivity Una volta avviata la LobbyActivity viene attivato il thread `serverConnection` che si occupa, per l'appunto, di collegarsi al server, ricavare tutte le informazioni utili allo scambio di messaggi (impostare `sender` e `receiver`, inviare il nome) e istanziare un nuovo oggetto della classe *custom* Server che contiene i seguenti attributi (ovviamente con relativi `getter` e `setter`):

```
InetAddress inetAddress;
Socket socket;
BufferedReader fromServer;
PrintWriter toServer;
```

Figura 8: Classe Server

Una volta ricevuto il messaggio di "`start`" da parte del server, posso procedere all'activity successiva.

CardSelectionActivity La CardSelectionActivity non è altro che un selettore di quante cartelle vuole il giocatore (minimo 1 e massimo 4), una volta premuto il pulsante partirà un messaggio verso il server.
A questo punto si attiva l'activity successiva.

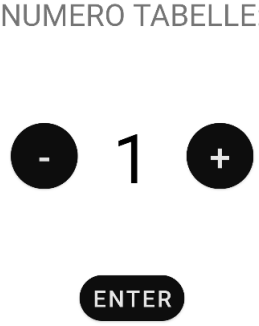


Figura 9: CardSelectionActivity

PreGameLobbyActivity La PreGameLobbyActivity è un'activity di transizione, si aspetta che tutti i client inseriscano il numero di cartelle che richiedono e si attende che il server le abbia create tutte e abbia il tempo per inviarle.
Una volta ricevute il messaggio di "`startloop`", il `Receiver` notifica questa activity e fa partire il Game Loop.

2.2.3 Playing Phase

Fase di gioco, client e server comunicano per sincronizzare l'uscita dei numeri.

GameLoopActivity Prima di tutto il client si occupa di ricevere tutte le cartelle di gioco e la stream dei numeri usciti, solo successivamente si occupa di renderizzare le tabelle. Estrae uno alla volta i numeri dalla stream, solo nel momento in cui viene notificato dal server, in modo tale da mantenere sempre la sincronizzazione.
Ad ogni estrazione viene aggiornata la grafica principale con la pallina estratta e la RecyclerView che rappresenta le cartelle.
Ho optato per sfondo grigio per i numeri ancora non usciti e sfondo blu per quelli gia usciti.
Ad ogni numero uscito viene avviata la funzione `findNumber` che si occupa di

controllare se il numero è presente oppure no nelle cartelle e, eventualmente, contrassegnare la sua `MatrixNumbers.flag` a `true`.

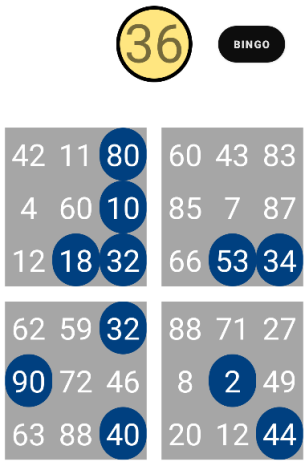


Figura 10: GameLoopActivity

Matrix La classe Matrix e la sua sotto-classe MatrixNumbers rappresentano il punto focale grazie al quale la rappresentazione delle cartelle nella RecyclerView può avvenire meglio.

```
public static class MatrixNumbers{
    private boolean flag;       //true -> exited, false -> not exited
    private int numero;
    private String numToString;

    public MatrixNumbers(int number, boolean flag){
        this.numero = number;
        this.flag = flag;
        this.numToString = Integer.toString(number);
    }
```

Figura 11: Sotto-Classe MatrixNumbers

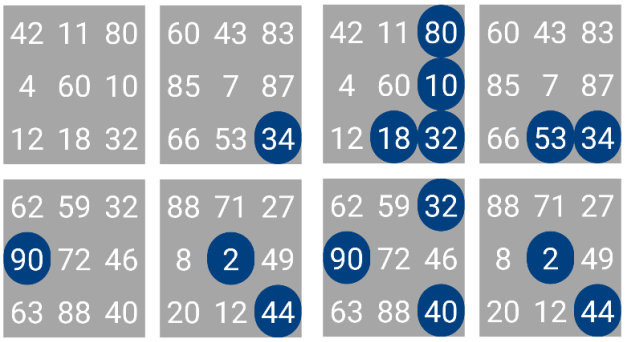
L'estrazione di un certo numero cambia il suo valore di `flag` in `true`, quindi, una volta nell'adapter, la cella corrispondente cambia colore dello sfondo. La classe Matrix non è altro che un array di MatrixNumbers.

```java
private final MatrixNumbers[] myMatrix;

public Matrix(MatrixNumbers[] myMatrix) {
    this.myMatrix = myMatrix;
}
```

Figura 12: Classe Matrix

RecyclerView e Adapter Nella parte grafica della GameLoopActivty è presente un'importantissima RecyclerView (`R.id.recycler`). Essa non è altro che la rappresentazione grafica delle matrici inviate dal server.
Per essere più precisi è un GridLayout di 2 colonne e 2 righe, ogni cella è occupata da un altro schema a griglia dove all'interno sono presenti i numeri. Ogni cella ha come colore standard quello grigio chiaro, succesivamente, a seconda dei numeri usciti, l'adapter fa cambiare colore allo sfondo del numero se esso possiede il `flag` uguale a `true`.
L'adapter (`MatrixAdapter`) viene richiamato goni volta che esce un nuovo numero e viene conluso il metodo `findNumber()` con esito positivo.



Bingo Button Il pulsante sulla destra è il pulsante da schiacciare nel momento del Bingo (quando viene completata una cartella).
Esso controlla che ci sia effettivamente una cartella completata e, in caso affermatico, invia un messaggio al server in cui lo comunica.

2.2.4 Conslusion Phase

La conclusion phase è la fase finale in cui viene attivata la WinnerActivity, vince il primo giocatore a finire una cartella e cliccare il pulsante "BINGO".

Il server si occupa di far apparire la grafica della vittoria e di inoltrare una notifica al client.

WinnerAvtivity Una volta arrivato il messaggio con il vincitore dal server, viene avviata la WinnerActivity. Essa non è altro che una semplicissima grafica con il nome del giocatore vincitore e un pulsante che permette di riavviare la partita (ripartendo dalla MainActivity, quindi quella di inserimento del nome).

Figura 13: WinnerActivity

3 Comunicazione Client - Server

La comunicazione Client - Server avviene tramite dei messaggi che identificano delle azioni in particolare a seconda del messaggio. Per questo motivo le comunicazioni devono essere regolate da un Receiver che filtra i messaggi e smista le azioni in modo da sincronizzare le parti della connessione stessa. Molto spesso i messaggi invocano dei metodi statici di notifica all'interno delle classi principali (come il GameLoop o la Lobby), per esempio vengono usati dei messaggi per notificare quando il client ha finito di renderizzare le cartelle in modo da poter far partire l'estrazione, oppure viene inviato un messaggio nel momento in cui viene premuto il pulsante "BINGO" e si invoca un intent che attiva la WinnerActivity.

I messaggi vengono anche utilizzati, ovviamente, per l'invio di dati come lo stream dei numeri estratti o le cartelle rappresentati i campi da gioco.
Lo stesso procedimento di notifica e invio di dati viene sfruttato, ovviamente, sia da client che da server in entrambi gli utilizzi.
Nel Server viene creato un `Thread Receiver` per ogni giocatore, esso viene anche utilizzato anche per memorizzare alcuni dati: nuome utente e il numero di cartelle richieste.