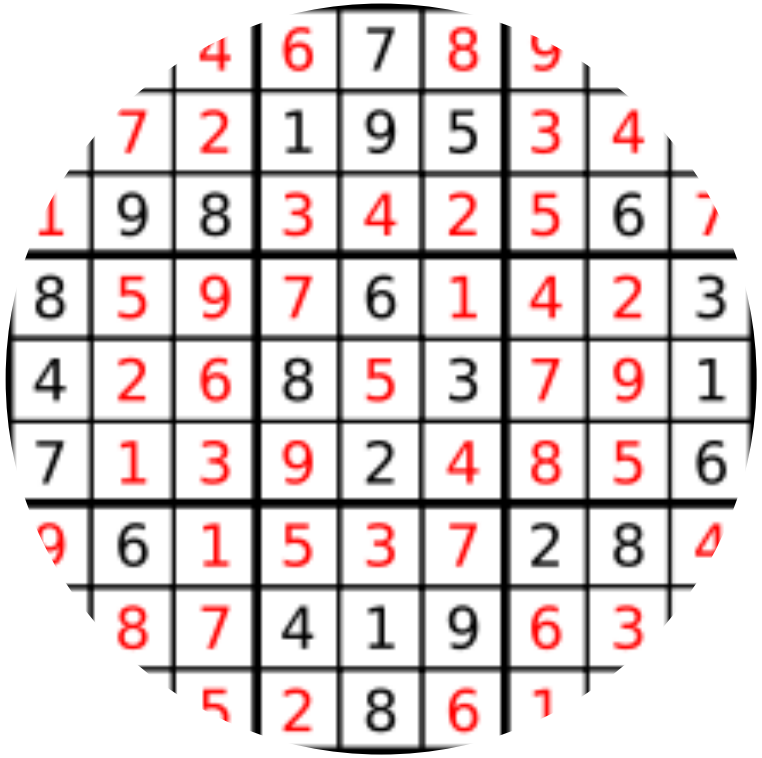
SUDOKU



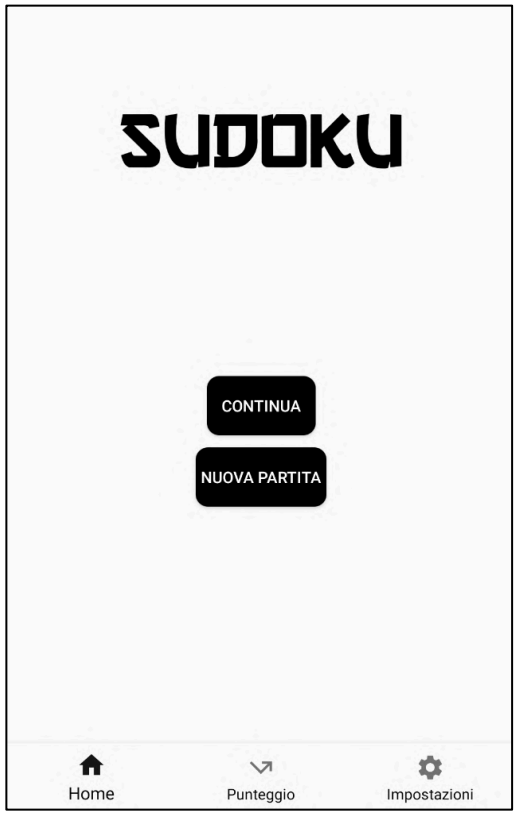
Adriano Brugnoni
Alessandro Fato
Simone Festa

Gruppo **BFF**

Struttura delle activity

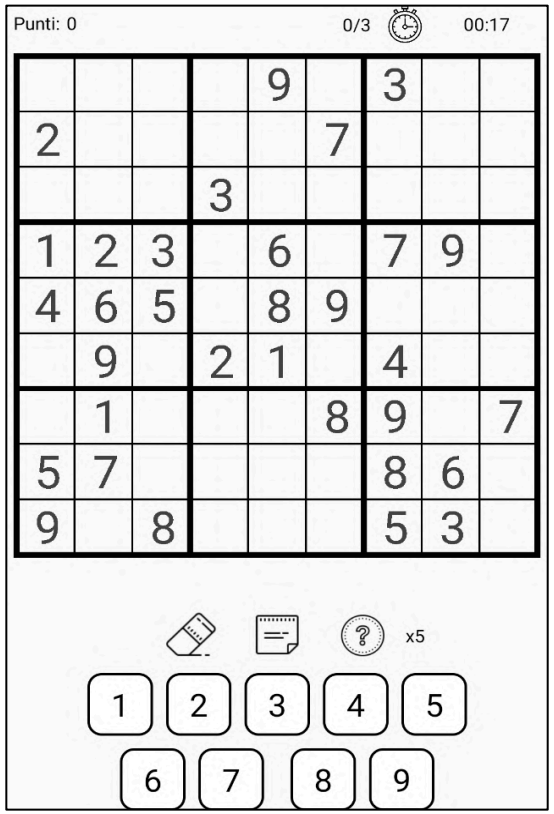
- MenuActivity

Composta da 3 Fragment



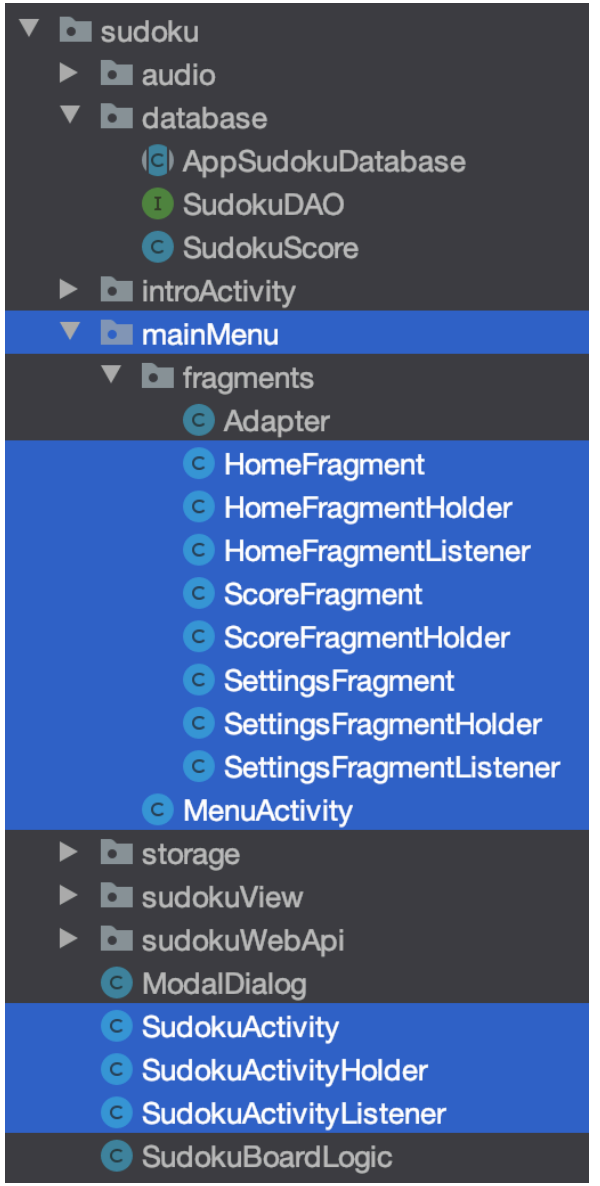
FragmentHome in foto

- SudokuActivity



In generale...

Le activity e i fragment sono caratterizzati da un holder e da un listener.



Dipendenze e permessi

Nel build.gradle viene aggiunto il riferimento a:

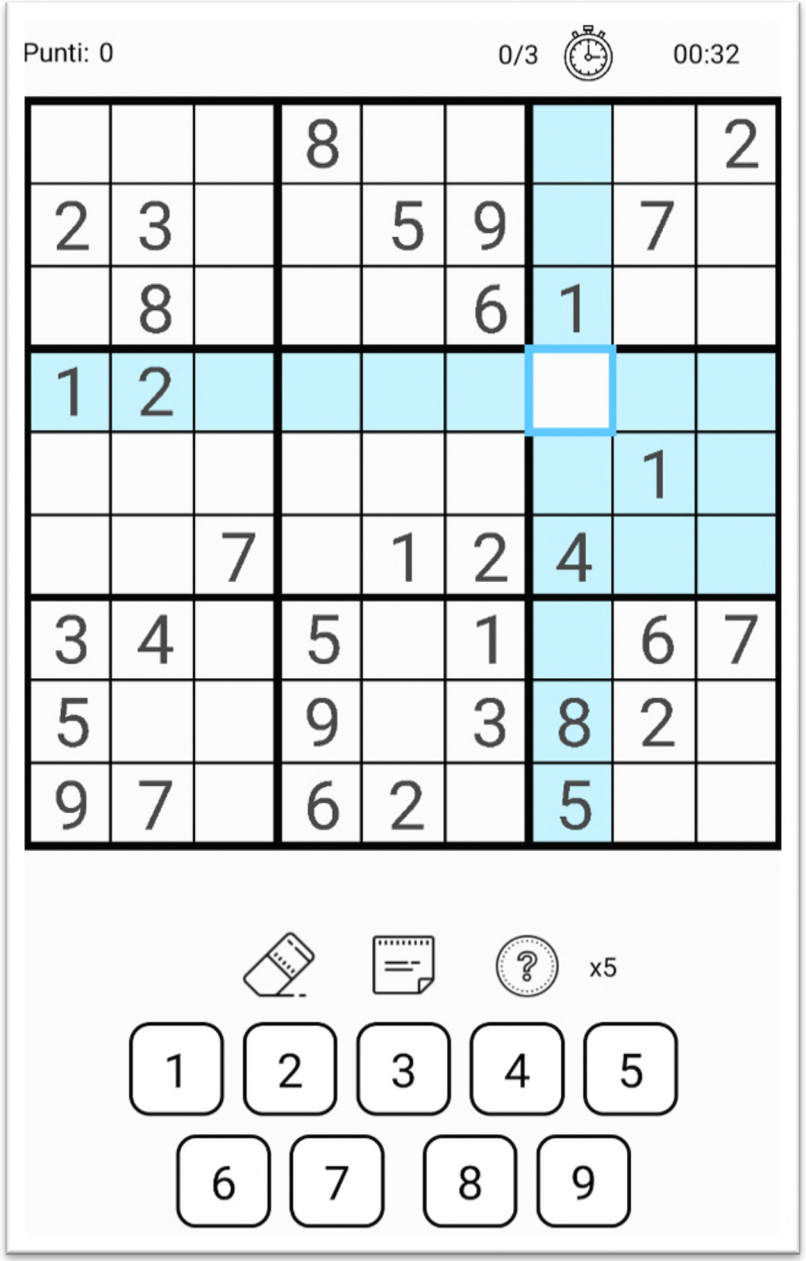
- **Room** per la gestione del database
- **Fragment** per gestire separatamente le porzioni di interfaccia che compongono l'applicazione.
- **Volley** per contattare le api del server per la generazione dei sudoku.

Nel manifest vengono richiesti l'utilizzo dello storage esterno e l'accesso a internet.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

SudokuActivity 1

- Gestisce l'esecuzione di una partita di sudoku
- Se chiamato onBackPressed viene salvato lo stato della partita attuale (riaccedibile tramite il bottone continua)
- È possibile:
  1. Inserire un numero in una cella per ottenere o perdere punti e, in caso, aumentare il contatore degli errori. A 3 errori la partita viene persa
  2. Abilitare/Disabilitare le note
  3. Cancellare le note da una cella
  4. Richiedere un aiuto in cambio di punti (max 5)
- Lo scorrere del timer diminuisce i punti



SudokuActivity 2

- La classe SudokuActivityHolder ha un oggetto di tipo SudokuView
- La SudokuView è la classe che gestisce la matrice del sudoku. Estende View

- Alla pressione dei bottoni (rilevata nel SudokuActivityListener) vengono effettuate operazioni direttamente sulla SudokuView

```
SudokuActivityHolder(SudokuActivity activity) {
    // ...
    sudokuView = activity.findViewById(R.id.sudokuView);
    // ...
}

void tryNumberInCell(int number) {
    if(useNote)
        sudokuView.setHint(number);
    else
        sudokuView.setNumber(number);
}
```

```
@Override
public void onClick(View v) {

    // ...

    // click on the number
    if(v.getId() == holder.getTvNumberOneId()) {
        holder.tryNumberInCell(1);
    }

    // ...
}
```

SudokuView 1

- Memorizza un oggetto di tipo SudokuLogic
- La classe SudokuLogic ha tutti i metodi e attributi per la gestione del sudoku

- È responsabile per la rappresentazione grafica della tabella del sudoku
  - onDraw: disegna la tabella con i numeri al suo interno (comprese le note). Lo stato della matrice del sudoku è memorizzato dentro l'oggetto SudokuLogic Il metodo calculatePainterTextSize assicura la corretta dimensione del testo nelle celle. Disegna inoltre il cursore che identifica la cella attualmente selezionata
  - onMeasure: modifica la dimensione della view per far sì che essa sia sempre un quadrato

- È responsabile di gestire il tocco sulla tabella per identificare la cella premuta
  - onTouch: identifica la cella premuta e ne salva le coordinate. Effettua un postInvalidate

  Le coordinate attuali (coordinatesTop, coordinatesLeft) vengono usate:
  1. nel metodo onDraw per disegnare il cursore
  2. in altri metodi per capire con quale cella interagire

SudokuView 2

- Mette a disposizione dei metodi pubblici chiamati dalla SudokuActivityHolder. Alcuni di questi sono:
  - setNumber: chiedi alla SudokuLogic di provare il numero nella cella attualmente

```
public void setNumber(int number) {
    sudoku.tryNumber(coordinatesTop, coordinatesLeft, number, SudokuLogic.MODE_GUESS);
    postInvalidate();
}
```

  - autoSetNumber: successivamente alla richiesta di un aiuto, auto completa una cella del sudoku

```
public void autoSetNumber() {
    int[] coordinates = sudoku.autoSetNumber();
    coordinatesTop = coordinates[0];
    coordinatesLeft = coordinates[1];
    postInvalidate();
}
```

SudokuView 3

- removeHints: rimuove tutte le note dalla cella attualmente selezionata

```
public void removeHints() {
    sudoku.deleteHints(coordinatesTop, coordinatesLeft);
    postInvalidate();
}
```

- setHint: aggiunge/rimuove una nota da una cella

```
public void setHint(int value) {
    sudoku.setHint(coordinatesTop, coordinatesLeft, value);
    postInvalidate();
}
```

- Tutte quante invocano il postInvalidate
- La logica di queste operazioni è implementata nella SudokuLogic

SudokuLogic 1

- Implementa le strutture dati per memorizzare la matrice del sudoku e delle note
- Implementa dei metodi per interagire con esse

```
private int[][] solvedMatrix;
private int[][] matrix;
private int[][] cellStatus;
private int[][][] hint;
```

- Alla sua istanziazione, il sudoku viene risolto e memorizzata la soluzione. Il thread SudokuSolveThread, data in input la matrice iniziale, risolve il sudoku. Alla sua terminazione (che potrebbe richiedere un tempo non banale) viene chiamato il metodo di callback onSudokuSolved nella classe SudokuLogic. Infatti, SudokuLogic implementa la classe SudokuSolveThread.CallBack

SudokuLogic 2

- La SudokuLogic implementa il pattern Observer
- Memorizza una lista di observer. Il metodo attach permette di aggiungere una classe che implementa SudokuCallBack alla lista degli observer
- L'inserimento di un numero nella tabella del sudoku può provocare un errore (quel numero in quella cella non è corretto) o un successo. Gli observer vengono notificati di questi eventi

```
// Observers
private List<SudokuCallBack> observers;
```

```
void attach(SudokuCallBack listener) {
    observers.add(listener);
}
```

```
public interface SudokuCallBack {
    void onSudokuError(int insertedValue, int cellTop, int cellLeft);
    void onSudokuSuccess(int insertedValue, int cellTop, int cellLeft, int remainingCells, int mode);
}
```

SudokuLogic 3

- La SudokuView è un osservatore della SudokuLogic
  - onSudokuError: illumina di rosso la cella in cui è avvenuto l'errore e riproduce un suono
  - onSudokuSuccess: riproduce un suono
- La SudokuActivity è un osservatore della SudokuLogic
  - onSudokuError: riduce i punti, incrementa il numero di errori e, in caso, chiude la partita
  - onSudokuSuccess: incrementa i punti (riducili se il successo è avvenuto grazie ad un consiglio) ed esci dalla partita se il sudoku è terminato

Nota: la grafica e il suono è gestita dalla SudokuView

SudokuLogic 4

- I metodi principali di logica sono:
  - setHint & deleteHint: inserisce/cancella le note da una cella
  - tryNumber: prova ad inserire il numero richiesto nella cella richiesta. È sufficiente confrontare il numero con la matrice risolta

```
void tryNumber(int top, int left, int number, int mode) {

    if(number == solvedMatrix[top][left]) {
        // set the number in the matrix ...
        // remove the hint useless hint ...
        notifyOnSudokuSuccess(number, top, left, getRemainingCells(), mode);
    }
    else {
        notifyOnSudokuError(number, top, left);
    }
}
```

SudokuLogic 5

- I metodi principali di logica sono:
  - autoSetNumber: viene riempita casualmente una cella del sudoku. È scelta casualmente tra la riga o la colonna o il quadrato con il numero minore di celle mancanti.

Sudoku web API 1

HomeFragmentListener - Metodo onClick

```
else if(v.getId() == holder.getBtnEasyId()) {
    holder.setViewInLoading(true);
    requestedDifficulty = VolleySudokuApi.Difficulty.EASY;
    sudokuApi.generateSudoku(requestedDifficulty);
}
else if(v.getId() == holder.getBtnMediumId()) {
    holder.setViewInLoading(true);
    requestedDifficulty = VolleySudokuApi.Difficulty.MEDIUM;
    sudokuApi.generateSudoku(requestedDifficulty);
}
else if(v.getId() == holder.getBtnHardId()) {
    holder.setViewInLoading(true);
    requestedDifficulty = VolleySudokuApi.Difficulty.HARD;
    sudokuApi.generateSudoku(requestedDifficulty);
}
```

Per generare il sudoku:

- Disabilito momentaneamente i bottoni
- Mostro la progress bar
- Contatto il server tramite Volley

Sudoku web API 2

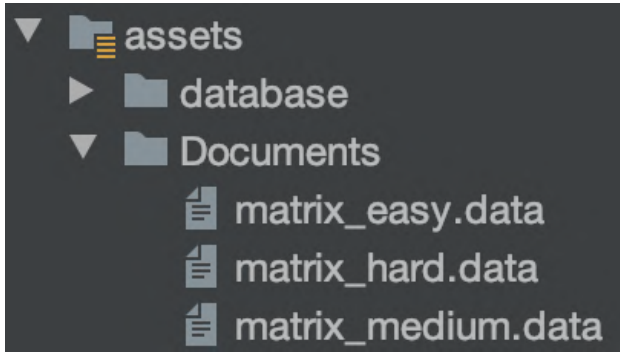
VolleySudokuApi

```
public void generateSudoku(Difficulty difficulty) {

    String url = "https://sugoku.herokuapp.com/board?difficulty=%s";
    url = String.format(url, difficulty.getName());
    apiCall(url);
}

private void apiCall(String url) {
    RequestQueue requestQueue;
    requestQueue = Volley.newRequestQueue(context);
    StringRequest stringRequest = new StringRequest(Request.Method.GET, url, responseListener, errorListener);
    requestQueue.add(stringRequest);
}
```

Sudoku web API 3

HomeFragmentListener gestirà il metodo di callback di Volley implementando Response.ErrorListener, Response.Listener<String>

```
public void onErrorResponse(VolleyError error) {
    loadGameFromStorage();
}
```




```
private void loadGameFromStorage() {

    String matrixToParse;
    String path;

    switch (requestedDifficulty) {
        case MEDIUM:
            path = MenuActivity.FILE_NAME_MATRIX_MEDIUM;
            break;
        case HARD:
            path = MenuActivity.FILE_NAME_MATRIX_HARD;
            break;
        default:
            path = MenuActivity.FILE_NAME_MATRIX_EASY;
            break;
    }

    // read all the game from file with the specified difficulty
    List<String> matrix = StoreManager.loadTextFileFromAsset(path,
            StoreManager.FileFormat.FORMAT_DATA,
            StoreManager.FileType.TYPE_DOCUMENT,
            activity);

    if(matrix == null || matrix.size() == 0) {
        holder.printShortToastMessage("Unable to create a new game");
        holder.setViewInLoading(false);
        return;
    }

    // extract one random game
    Random random = new Random();
    int randomValue = random.nextInt(matrix.size());
    matrixToParse = matrix.get(randomValue);

    startNewGame(matrixToParse);
}
```

Sudoku web API 4

Il metodo getGameMatricFromJson restituisce, dato in input il file json restituito dal server, una stringa che rappresenta lo stato del sudoku

```
public void onResponse(String response) {

    String board;
    try {
        // create the json object from the string in response that represents a json file
        JSONObject jsonObject = new JSONObject(response);
        // read the array that contains the Board
        board = jsonObject.getJSONArray( name: "board").toString();

        String matrixToParse = SudokuLogic.getGameMatrixFromJson(board,  sudokuDim: 9);

        startNewGame(matrixToParse);

    } catch(JSONException e) {
        e.printStackTrace();
        loadGameFromStorage();
    }
}
```

Gestione dei fragment

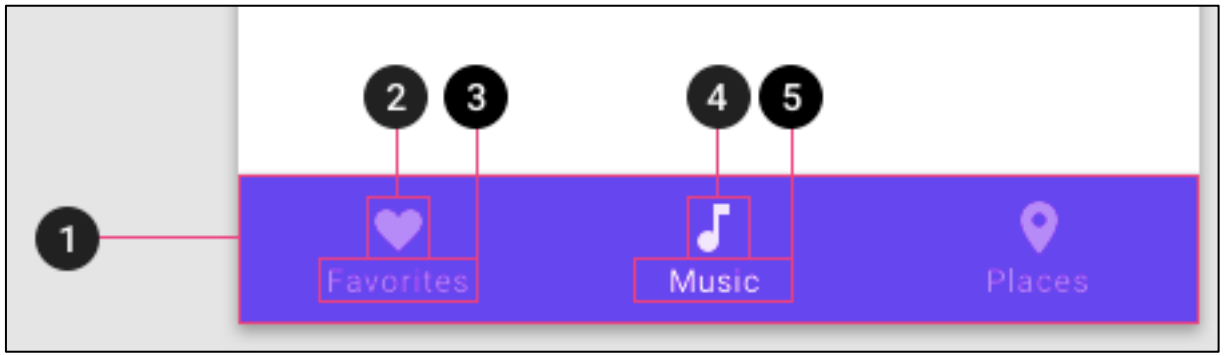
- La loro implementazione è una diretta conseguenza dell'utilizzo di BottomNavigationView.

- Consente di switchare tra le varie view (fino ad un massimo di 5).

- E' altamente personalizzabile e coerente con il Material Design. Può includere elementi statici (icone, testo) ed elementi dinamici (badge con numero di notifiche, o un semplice pallino).

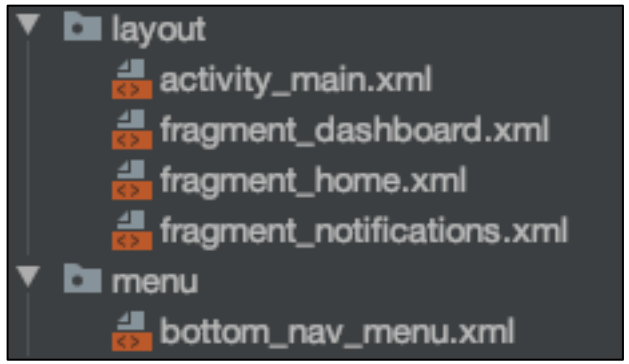
Componenti della navbar

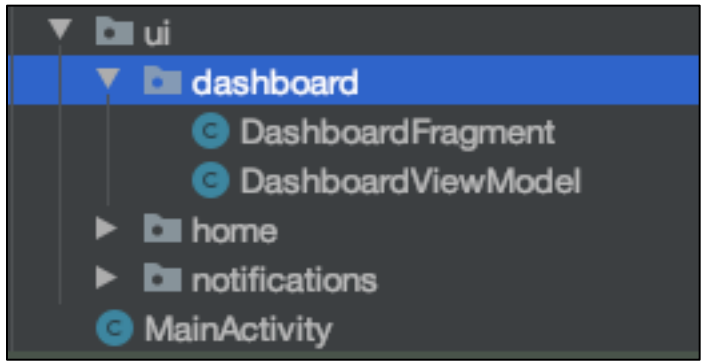
- 1) Container : include tutte le componenti della navbar.
- 2) e 3) Icona e testo inattivi: elementi statici. Se non attiva, non ci troviamo nel suo fragment associato.
- 4) e 5) Icona e testo attivi: elementi statici. Presente una colorazione diversa, ed evidenzia il fatto che ci troviamo nel fragment associato.
- Elementi attivi e non attivi sono gestiti automaticamente!



Realizzazione della navbar

- Android Studio crea automaticamente i file xml della navbar, dei fragment e della main activity.
- Crea anche, nel package «ui» le classi Fragment e viewModel dei vari elementi presenti nella navbar.





Realizzazione della navbar 2

- Abbiamo anche, nel package «navigation», un file chiamato mobile_navigation.xml

- Rappresenta un «grafo» di tutte le destinazioni, e di tutte le azioni che creano interazioni.

- Nel nostro caso le interazioni non avvengono direttamente tra fragment, questo perché il passaggio tra le varie interfacce è gestito dalla navbar, e non da elementi presenti nei fragment (e.g. bottoni).

Come funziona?

- La classe MenuActivity implementa BottomNavigationView.OnNavigationItemSelectedListener

- Dentro onCreate viene istanziata la classe BottomNavigationView: essa viene associata alla controparte grafica tramite findViewById.

- Viene anche richiamato il metodo setOnNavigationItemSelectedListener.

```
setContentView(R.layout.activity_menu);
BottomNavigationView navView = findViewById(R.id.nav_view);

navView.setOnNavigationItemSelectedListener(this);
getSupportFragmentManager().beginTransaction().replace(R.id.nav_host_fragment, new HomeFragment( activity: this)).commit();
```

Come funziona ? 2

- Nella nostra MenuActivity richiamiamo getSupportFragmentManager che ritorna un FragmentManager.
- Richiamiamo anche beginTransaction(), che crea FragmentTransaction.
- Mediante add, il fragment viene aggiunto.
- Con il commit rendiamo effettivi questi cambiamenti.

```
getSupportFragmentManager().beginTransaction().add(R.id.nav_host_fragment, new HomeFragment( activity: this)).commit();
```

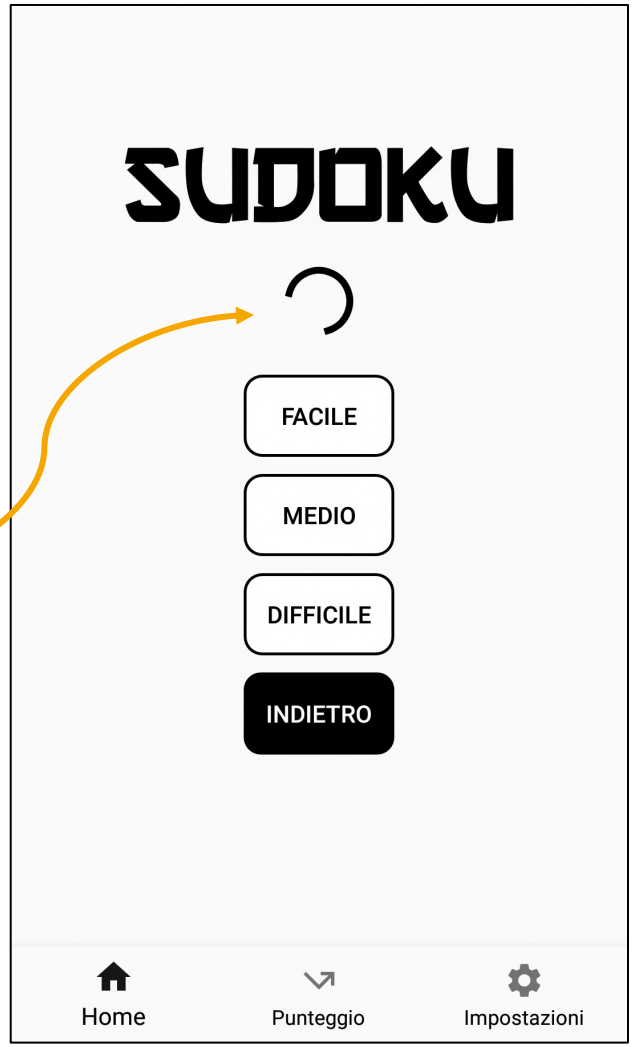
Passaggio tra fragment

- Con il metodo onNavigationItemSelected in MenuActivity possiamo switchare tra le varie componenti dell'applicazione.
- A seconda dell'icona premuta sulla navbar, si viene reindirizzati al fragment corrispondente.
- Le modifiche si rendono effettive richiamando il SupportFragmentManager.
- Poiché ci stiamo muovendo tra fragment, useremo replace al posto di add!

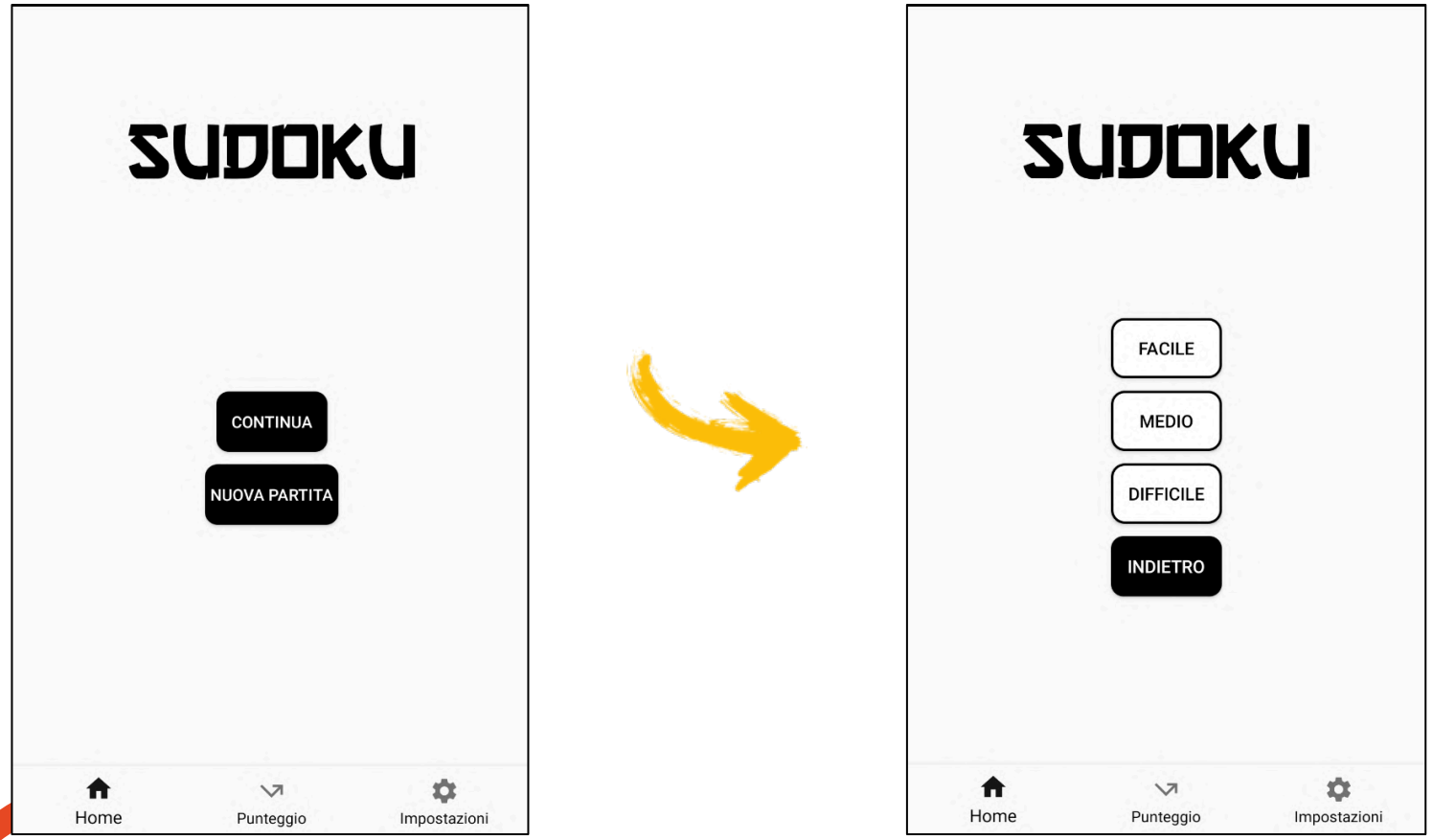
```
@Override
public boolean onNavigationItemSelected(@NonNull MenuItem item) {
    Fragment selectedFragment;
    prepareSoundAndPlay(controllerAudio);
    switch (item.getItemId()) {
        case R.id.navigation_home:
            selectedFragment = new HomeFragment( activity: this);
            break;
        case R.id.navigation_settings:
            selectedFragment = new SettingsFragment( activity: this);
            break;
        case R.id.navigation_score:
            selectedFragment = new ScoreFragment( activity: this);
            break;
        default:
            return false;
    }
    getSupportFragmentManager().beginTransaction().replace(R.id.nav_host_fragment, selectedFragment).commit();
    return true;
}
```

1° Fragment: HomeFragment

- Nell'holder associamo i vari bottoni alle loro controparti grafiche.
- I bottoni per iniziare/continuare una partita, e per scegliere la difficoltà sono associati ad un listener.
- Quando scegliamo la difficoltà, viene visualizzata una barra di progresso.
- Quando la matrice Sudoku è pronta, la barra viene disattivata e si passa alla SudokuActivity.
- I layout del Menu (carica partita/nuova partita) e della selezione della difficoltà (facile, medio, difficile) sono mutuamente esclusivi.



1° Fragment: HomeFragment



- L'utente può caricare l'ultima partita salvata (se presente).
- L'utente può iniziare una nuova partita generata da una server api. Se la connessione a internet non è presente (o insufficiente) il sudoku viene estratto casualmente dalla memoria locale.

Caricamento partita

- Alla pressione del bottone "continua" (HomeFragmentListener) viene caricato il file contenente la partita salvata
- Nella startNewGame viene passato come Intent alla SudokuActivity
- Nella onCreate della SudokuActivity vengono impostati correttamente il timer, i punti, gli errori, il numero di aiuti rimasti
- La SudokuLogic viene inizializzata tramite setupMatrix. Riceve come parametro la matrice sotto forma di stringa e ne ricava il valore delle celle ed il loro stato (e risolve il sudoku)

```
public void onClick(View v) {

    // ...

    if(v.getId() == holder.getBtnContinueId()) {
        String matrixToParse;

        // get the saved game from internal storage
        List<String> savedGame = StoreManager.loadTextFile(SudokuActivity.SAVE_FILE_NAME,
                StoreManager.FileFormat.FORMAT_DATA,
                StoreManager.FileType.TYPE_DOCUMENT, activity);

        if(savedGame == null) {
            holder.printShortToastMessage("No game save present");
            return;
        }

        matrixToParse = savedGame.get(0);

        startNewGame(matrixToParse);
    }

    // ...
}
```

Salvataggio partita

- Il metodo onBackPressed della SudokuActivity memorizza lo stato della partita su file

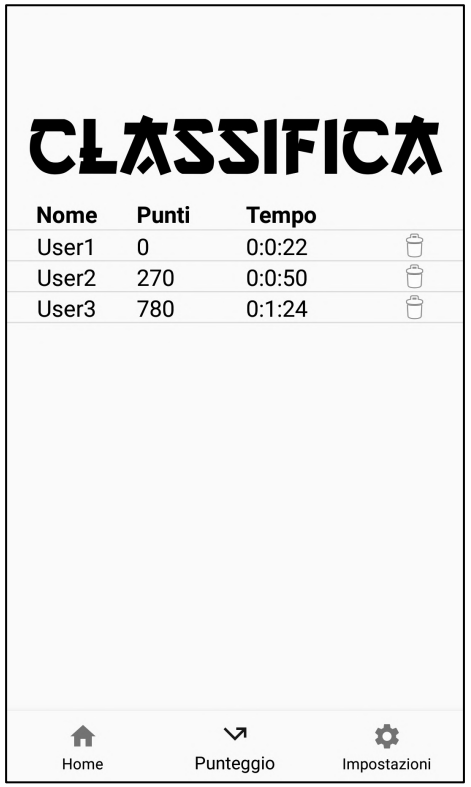
```
String game = holder.getActualGameMatrix();
boolean saved = StoreManager.storeTextFile(SAVE_FILE_NAME, game,
        StoreManager.FileFormat.FORMAT_DATA,
        StoreManager.FileType.TYPE_DOCUMENT, activity: this);
```

- Viene invocato il metodo getActualGameStatus della classe SudokuLogic. Tale metodo, preso lo stato del sudoku (valori e stato delle celle), costruisce una stringa. Vengono aggiunti successivamente il valore del timer, i punti, gli errori, il numero di aiuti rimasti.
- La classe StoreManager aiuta nella scrittura e lettura di file dalla memoria del device
- Un salvataggio possibile è:

  - 00;00;00;21;00;81;00;00;31;00;00;00;00;00;00;00;00;41;00;00;11;00;91;00;51;00;21;00;41;31;00;00;81;00;00;31;00;00;81;91;11;00;00;21;00;00;91;71;21;41;31;00;00;51;31;11;00;41;21;00;71;81;71;41;00;00;00;31;51;21;00;00;00;00;00;00;00;61;31;00;&;235;;;;;;;;;;;;;;;;;;;;;;;;;;;56;;3;;;;;;;;;;;;;;&0:0:39&0&0&5

2° Fragment: ScoreFragment

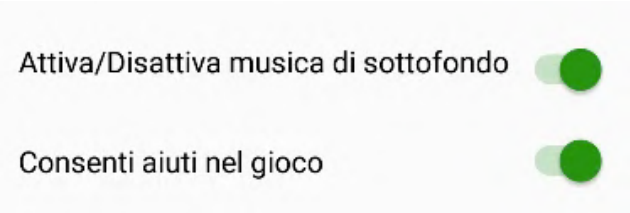
- Il font del titolo è go3v2.tff (Lo stesso di Fruit Ninja)
- Vengono rappresentati Nome, Punti e Tempo.
- Con una pressione prolungata è possibile eliminare definitivamente un risultato.
- Presenta un holder, nel quale è definita una RecyclerView e un collegamento al Db e Adapter. Tali aspetti, legati alla gestione del Db, saranno visualizzati nelle slide successive.



3° Fragment: SettingsFragment

- Presenta due switch, i quali consentono di attivare/disattivare consigli e musica in sottofondo.

- A seconda dello stato dell'opzione, lo switch cambia anche colore!



- Basta aggiungere in styles.xml queste poche righe di codice:

```
<style name="SwitchTheme" parent="Theme.AppCompat.Light">
    <item name="android:colorControlActivated">#148E13</item>
</style>
```

3° Fragment: SettingsFragment

- Le modifiche sono persistenti!
- Quando si preme su uno switch appare un toast che ci informa della nuova impostazione.
- Il controllo viene eseguito mediante «isChecked», presente nel metodo onCheckedChanged.
- In questo metodo viene istanziato un oggetto di tipo PreferencesManager.

3° Fragment: SettingsFragment


```
if(buttonView.getId() == holder.getSwitchSoundId()) {

    if(isChecked){
        preferences.set(PreferencesManager.KEY_SOUND, true);
        msg = "Music on";
    }
    else{
        preferences.set(PreferencesManager.KEY_SOUND, false);
        msg = "Music off";
    }

    Toast.makeText(activity.getApplicationContext(), msg,Toast.LENGTH_SHORT).show();
```

Memorizzazione delle preferenze

- Facciamo riferimento alla classe PreferencesManager.

- Abbiamo intenzione di memorizzare pochi valori chiave riferiti alle impostazioni dell'applicazione :

```
public final static String KEY_SOUND = "sound";
public final static String KEY_HELP = "help";
```

- Possiamo utilizzare l'interfaccia SharedPreferences, la quale punta ad un file contenente valori chiavi e metodi per l'accesso in lettura e scrittura.

Memorizzazione delle preferenze - 2

- Troviamo metodi get e set, in cui passiamo la key.
- Con il metodo set, richiamo prima SharedPreferences e poi SharedPreferences Editor per modificare le key.
- A seconda della key passata e del suo valore, l'editor inserisce nelle preferenze il file booleano, applicando poi i cambiamenti.
- Con il metodo get, richiamiamo SharedPreferences che ritorna il valore della chiave passata come parametro.

Memorizzazione delle preferenze - 3

```
public void set(String key, Object value) {

    SharedPreferences sharedPreferences = context.getSharedPreferences( name: "Preferences", Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();

    switch (key) {
        case KEY_SOUND:
            editor.putBoolean(KEY_SOUND, (boolean) value);
            break;
        case KEY_HELP:
            editor.putBoolean(KEY_HELP, (boolean) value);
            break;
    }

    editor.apply();
}
```

Accessibile solo da applicazioni che condividono stesso user ID o da applicazioni chiamanti.

Nome del file dove vengono memorizzate le key

Dove vengono memorizzati?

- Le key vengono memorizzate nel percorso :

  data/data/it.bff.sudoku/shared_prefs


```xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="help" value="true" />
    <boolean name="sound" value="true" />
</map>
```

Soluzione Sudoku

Per poter comunicare all'utente se il suo inserimento è valido o meno l'app risolve preliminarmente il Sudoku.
Teorema e proposizioni a sostegno dell'algoritmo implementato:

Teorema:

Dato un Sudoku 9x9, esso ha al più una soluzione se presenta almeno 8 caselle riempite.
Più formalmente un grafo ottenuto da un Sudoku è al più 9-colorabile se presenta almeno già 8 vertici colorati.
Se ne presenta al più 7 allora il grafo ammette almeno due 9-colorazioni.

Dimostrazione:

(http://www.ams.org/notices/200706/tx070600708p.pdf)

Soluzione Sudoku

Proposizione 1:

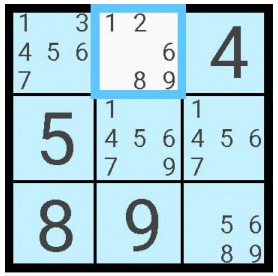
Dato un qualsiasi quadrato Q $3x3$ composto da celle c_{ij} di un Sudoku n×n e un numero $k \in [1,9] \epsilon \mathbb{N}$, se $!\exists c_{ij} \in Q : k$ è $un\ numero\ valido$ allora quella cella avrà k come valore nella soluzione.

Proposizione 2:

Dato un qualsiasi riga R composta da celle c_{ij} di un Sudoku n×n e un numero $k \in [1,9] \epsilon \mathbb{N}$, se $!\exists c_{ij} \in R : k$ è $un\ numero\ valido$ allora quella cella avrà k come valore nella soluzione.

Proposizione 3:

Dato un qualsiasi colonna C composto da celle c_{ij} di un Sudoku n×n e un numero $k \in [1,9] \epsilon \mathbb{N}$, se $!\exists c_{ij} \in C : k$ è $un\ numero\ valido$ allora quella cella avrà k come valore nella soluzione.



Soluzione Sudoku

Proposizione 1:

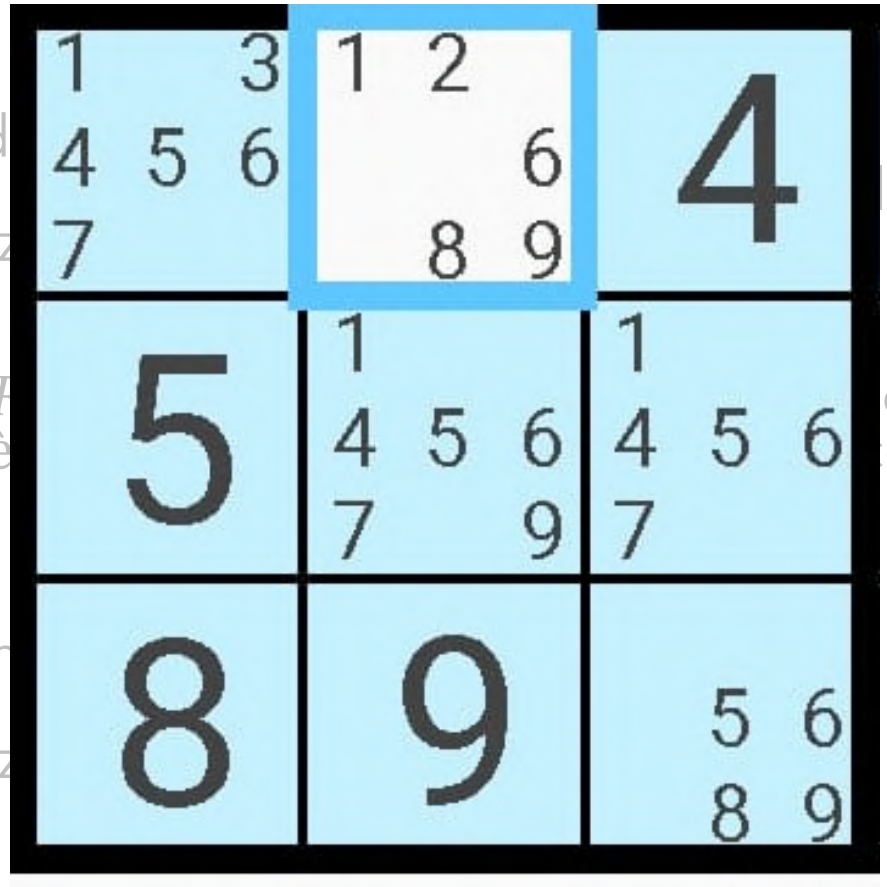
Dato un qualsiasi quad di un Sudoku n×n e un numero $k \in [1,9]\epsilon\mathbb{N}$, se llora quella cella avrà k come valore nella soluz

Proposizione 2:

Dato un qualsiasi riga R oku n×n e un numero $k \in [1,9]\epsilon\mathbb{N}$, se $!\exists c_{ij} \in R : k$ è ella avrà k come valore nella soluzione.

Proposizione 3:

Dato un qualsiasi colon Sudoku n×n e un numero $k \in [1,9]\epsilon\mathbb{N}$, se llora quella cella avrà k come valore nella soluz



Zoom sulla funzione `solveBrute(int[][] matrix, int n)`

Verifica se vi è uno spazio vuoto.
Se non c'è uno spazio vuoto allora il sudoku è risolto.
Altrimenti, una volta individuato procedi.

```
boolean solve(int[][] matrix, int n)
{
    int row = -1;
    int col = -1;
    boolean flag = true;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (matrix[i][j] == 0)
            {
                row = i;
                col = j;
                flag = false;
                break;
            }
        }
        if (!flag)
            break;
    }
    if (flag)
        return true;
    for (int k = 1; k <= n; k++)
    {
        if (check(matrix, row, col, k))
        {
            matrix[row][col] = k;
            if (solve(matrix, n))
                return true;
            else
                matrix[row][col] = 0;
        }
    }
    return false;
}
```

```java
private boolean check(int[][] matrix, int row, int col, int num)
{
    int rad = (int) Math.sqrt(matrix.length);
    for (int i = 0; i < matrix.length; i++)
    {
        if (matrix[row][i] == num || matrix[i][col] == num)
            return false;
    }
    for (int r = row - row % rad; r < row - row % rad + rad; r++)
    {
        for (int d = col - col % rad; d < col - col % rad + rad; d++)
        {
            if (matrix[r][d] == num)
                return false;
        }
    }
    return true;
}
```

Abbiamo individuato la casella su cui lavorare.
Proviamo gli n numeri del nostro Sudoku $n \times n$, ovvero verifichiamo la loro validità nel Sudoku ogni volta.
Se è temporaneamente valido, richiamare ricorsivamente la funzione solve fino al completamento.

```java
boolean solve(int[][] matrix, int n)
{
    int row = -1;
    int col = -1;
    boolean flag = true;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (matrix[i][j] == 0)
            {
                row = i;
                col = j;
                flag = false;
                break;
            }
        }
        if (!flag)
            break;
    }
    if (flag)
        return true;
    for (int k = 1; k <= n; k++)
    {
        if (check(matrix, row, col, k))
        {
            matrix[row][col] = k;
            if (solve(matrix, n))
                return true;
            else
                matrix[row][col] = 0;
        }
    }
    return false;
}
```

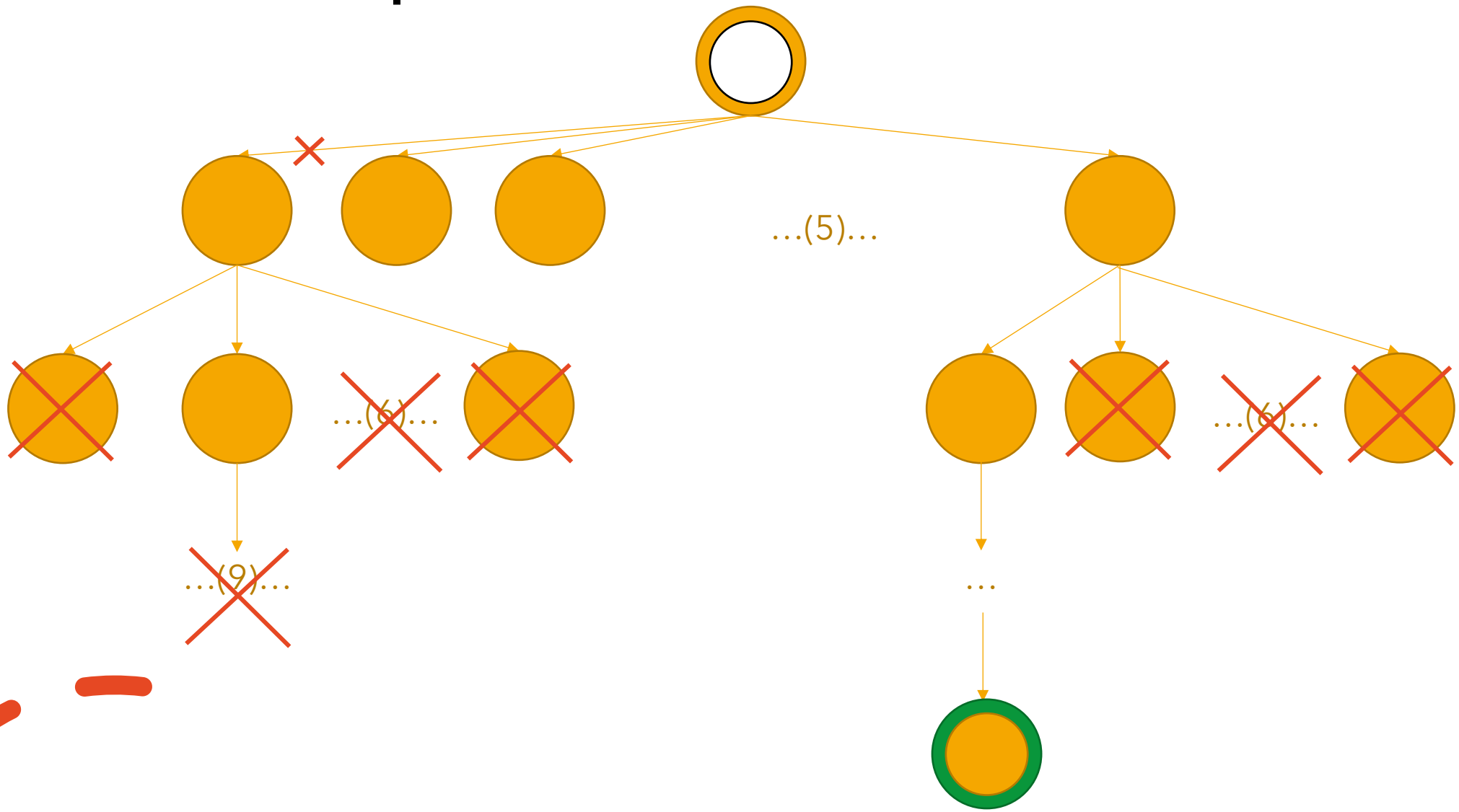
Zoom sulla funzione check()

Controllo sulla riga e colonna corrente

Controllo sul quadrato corrente

```
private boolean check(int[][] matrix, int row, int col, int num)
{
    int rad = (int) Math.sqrt(matrix.length);
    for (int i = 0; i < matrix.length; i++)
    {
        if (matrix[row][i] == num || matrix[i][col] == num)
            return false;
    }
    for (int r = row - row % rad; r < row - row % rad + rad; r++)
    {
        for (int d = col - col % rad; d < col - col % rad + rad; d++)
        {
            if (matrix[r][d] == num)
                return false;
        }
    }
    return true;
}
```

SudokuGraph



Panoramica gestione Audio

- Necessità:
  - Bassa latenza: buon feedback all'utente;
  - Possibilità di gestire dimensione generiche di file audio;
  - Controllo sull'integrità dei file audio.

- Soluzione:
  - Uso di *AudioTrack*: libreria con completa gestione per file .wav e ottima latenza per applicazioni non strettamente musicali;
  - Implementazione di un *parser WAV*(vedere ……).

AudioTrack

- Creazione oggetto AudioTrack:

```
new AudioTrack(
        int streamType,
        int sampleRateInHz,
        int channelConfig,
        int audioFormat,
        int bufferSizeInBytes,
        int mode,
        int session);
```

- STREAM_MUSIC
- STREAM_VOICE
- STREAM_ALARM
- …

- Mono
- Stereo
- …

Tipologia di modulazione.
Caso supportato dall'app:
Modulazione ad Impulsi Codificati (PCM)

- Static
- Stream

AudioTrack – in dettaglio(1) (int streamType)

Il parametro `int streamType` ci permette quindi di informare il sistema operativo riguardo la tipologia di flusso audio che stiamo immettendo.

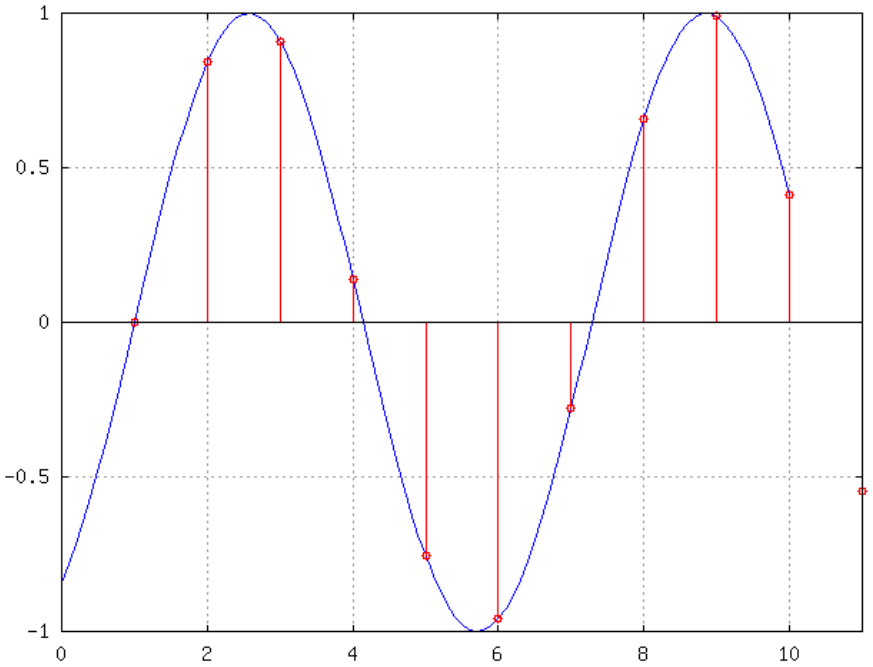
Attraverso questa informazione il S.O. può gestire l'audio di ogni singola tipologia di flusso.

Ad esempio:
Poter mutare il canale musicale, ma tener attivo il canale audio delle sveglie.

```
new AudioTrack(
        int streamType,

        int sampleRateInHz,

        int channelConfig,

        int audioFormat,

        int bufferSizeInBytes,

        int mode,

        int session);
```

AudioTrack – in dettaglio(2) (int sampleRateInHz)

E' possibile inserire la frequenza di campionamento del nostro file audio mediante il parametro `int sampleRateInHz`.
E' possibile specificare la tipologia di modulazione mediante il parametro `int audioFormat`.




```
new AudioTrack(
        int streamType,

        int sampleRateInHz,

        int channelConfig,

        int audioFormat,

        int bufferSizeInBytes,

        int mode,

        int session);
```

AudioTrack – in dettaglio(3) (int mode)

- Modalità streaming: possibilità di potere scrivere su uno stream continuo, utile in caso di audio generato a run-time o casistiche in cui non si vuole usare eccessivamente la memoria.

- Modalità statica: utilizzata per situazioni in cui si necessita bassa latenza(ad es. giochi, app interattive, …)

L'ultima in questione è la modalità a noi comoda.

Buon compromesso per riprodurre sia musica in background che effetti audio durante l'esecuzione dell'intera applicazione.

```
new AudioTrack(
        int streamType,

        int sampleRateInHz,

        int channelConfig,

        int audioFormat,

        int bufferSizeInBytes,

        int mode,

        int session);
```

AudioTrack – in dettaglio(4) (int session)

- E' possibile dividere in sessioni i nostri oggetti Audiotrack.
- Attraverso esse è possibile identificare ogni singolo flusso audio mediante un listener che scandisce per sessione.

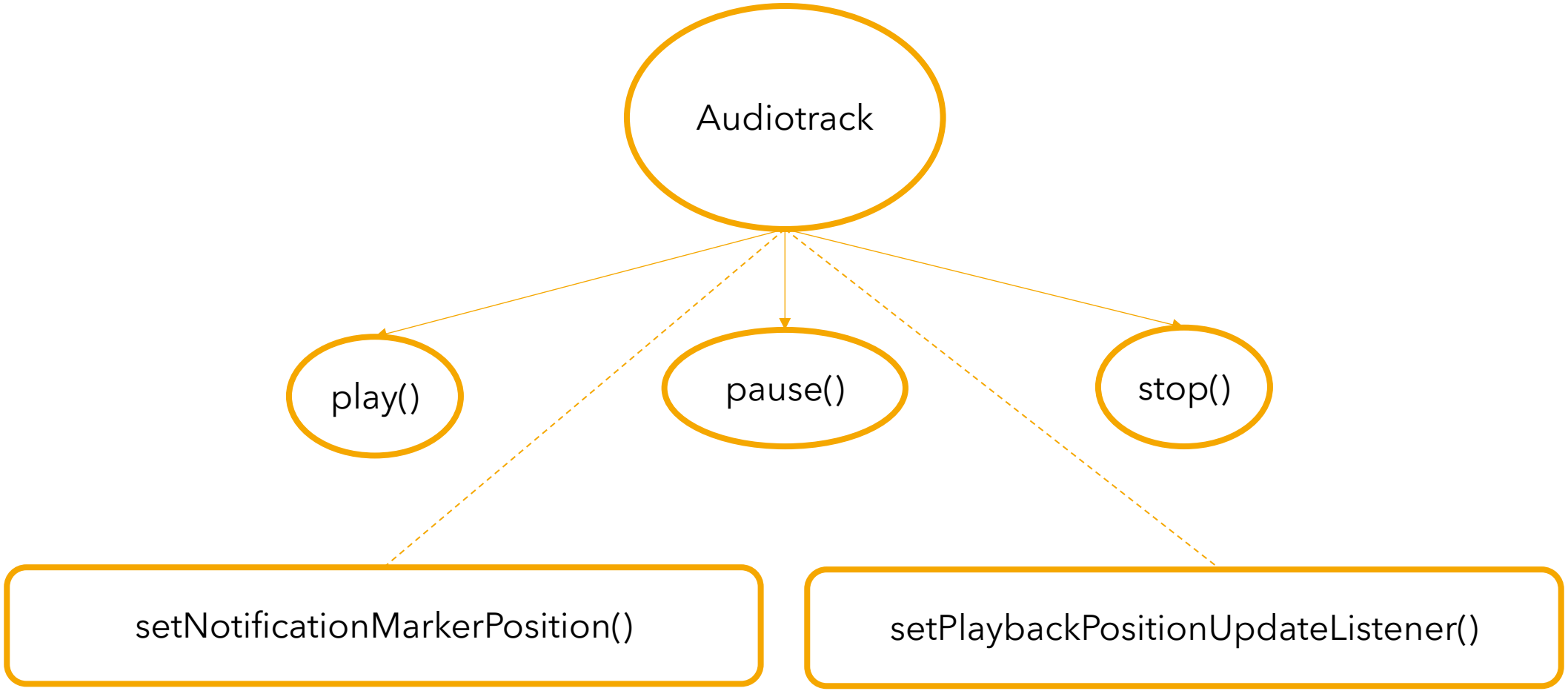
```
new AudioTrack(
        int streamType,

        int sampleRateInHz,

        int channelConfig,

        int audioFormat,

        int bufferSizeInBytes,

        int mode,

        int session);
```

Controller Audio

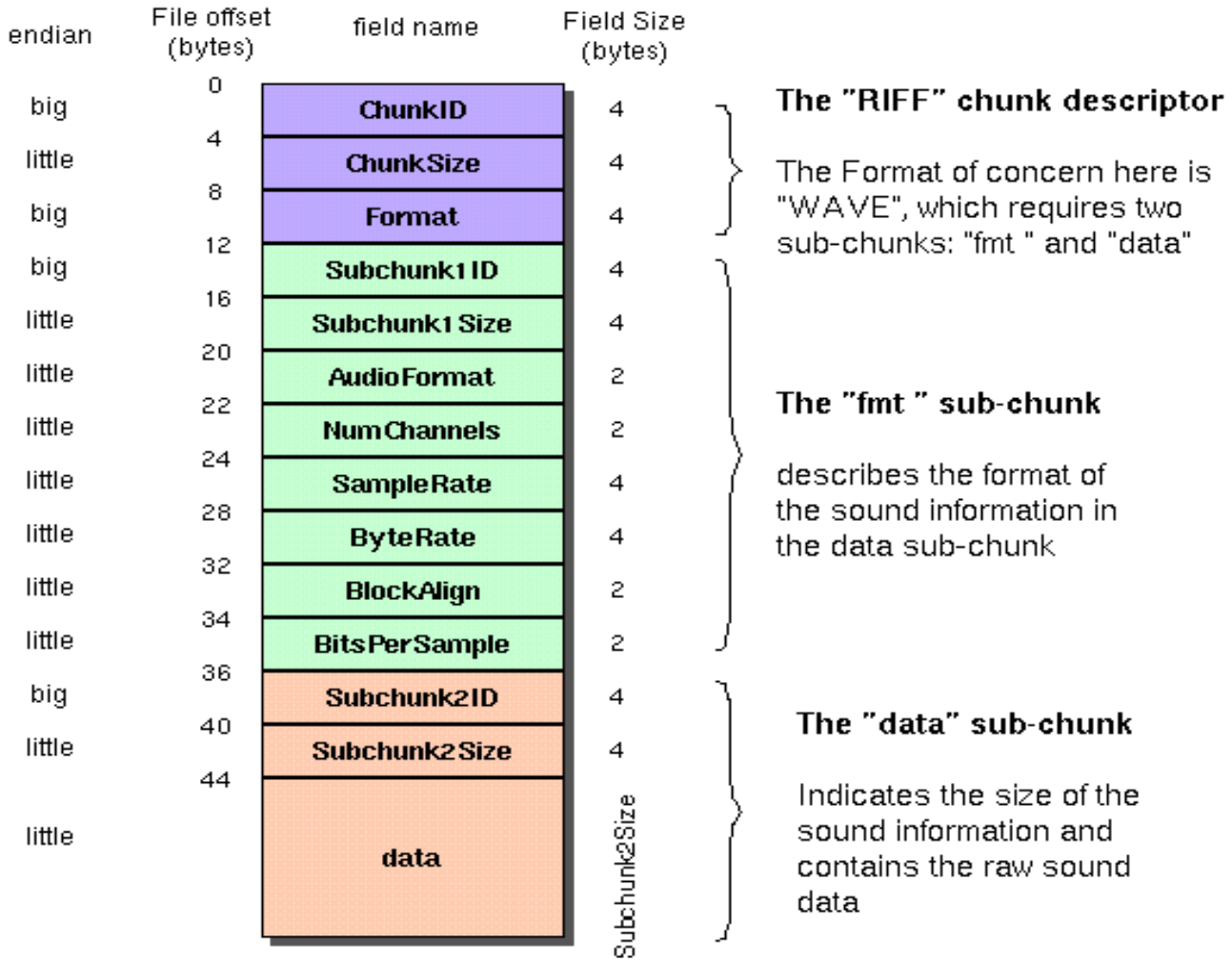
- In game possiamo notare l'introduzione di suoni d'errore, di sottofondo e di corretto inserimento.
- Tramite la classe **WAVFILE** otteniamo le informazioni dei file.wav
- In ControllerAudio troviamo la gestione dei suoni, quali attivazione, stop, ripetizioni ecc…
- Ogni traccia avviata avrà il proprio listener associato, pronto a catturare il marcatore ad essa associato.

```
public void prepareSoundAndPlay() throws IOException
{
    if(!isPrepared)
    {
        byte[] byteWav = wavFile.createWavItem();
        wav = new short[(wavFile.getDimension() - 44) / 2];

ByteBuffer.wrap(byteWav).order(ByteOrder.LITTLE_ENDIAN).
        asShortBuffer().get(wav);
        isPrepared=true;
    }
    playShortModeStatic(wav);
}

public void stopAndRelase()
{
    audioTrack.stop();
    audioTrack.release();
    audioTrack=null;
}
public void pause(){ audioTrack.pause(); }
public void resume() { audioTrack.play(); }

public boolean isInitialized()
{
    if(audioTrack!=null)
        return true;
    else
        return false;
}
```

Operazioni Audiotrack



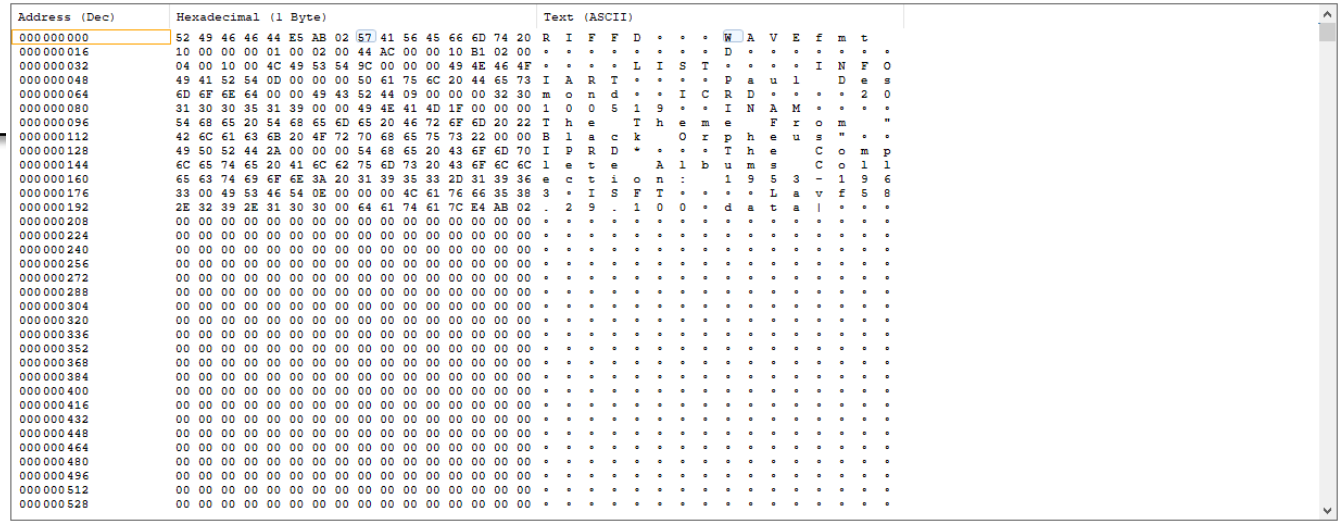
Struttura WAV



Classe WavFile – controllo header(1)

Ogni file .wav viene mappato in un oggetto WavFile.
Di ogni file mappato ne viene controllato l'inizio dell' header:

```
private boolean checkWavFile(byte[] buffer)
{
    if(buffer[0]=='R' && buffer[1]=='I' && buffer[2]=='F' && buffer[3]=='F' && buffer[8]=='W' && buffer[9]=='A' && buffer[10]=='V' &&
buffer[11]=='E')
        return true;
    else
        return false;
}
```



Address (Dec)	Hexadecimal (1 Byte)	Text (ASCII)
000000000	52 49 46 46 44 E5 AB 02 57 41 56 45 66 6D 74 20	R I F F D • • • W A V E f m t
000000016	10 00 00 00 01 00 02 00 44 AC 00 00 10 B1 02 00	• • • • • • • • D • • • • • • •
000000032	04 00 10 00 4C 49 53 54 9C 00 00 00 49 4E 46 4F	• • • • L I S T • • • • I N F O
000000048	49 41 52 54 0D 00 00 00 50 61 75 6C 20 44 65 73	I A R T • • • • P a u l D e s
000000064	6D 6F 6E 64 00 00 49 43 52 44 09 00 00 00 32 30	m o n d • • I C R D • • • • 2 0
000000080	31 30 30 35 31 39 00 00 49 4E 41 4D 1F 00 00 00	1 0 0 5 1 9 • • I N A M • • • •
000000096	54 68 65 20 54 68 65 6D 65 20 46 72 6F 6D 20 22	T h e T h e m e F r o m "
000000112	42 6C 61 63 6B 20 4F 72 70 68 65 75 73 22 00 00	B l a c k O r p h e u s " • •
000000128	49 50 52 44 2A 00 00 00 54 68 65 20 43 6F 6D 70	I P R D * • • • T h e C o m p
000000144	6C 65 74 65 20 41 6C 62 75 6D 73 20 43 6F 6C 6C	l e t e A l b u m s C o l l
000000160	65 63 74 69 6F 6E 3A 20 31 39 35 33 2D 31 39 36	e c t i o n : 1 9 5 3 - 1 9 6
000000176	33 00 49 53 46 54 0E 00 00 00 4C 61 76 66 35 38	3 • I S F T • • • • L a v f 5 8
000000192	2E 32 39 2E 31 30 30 00 64 61 74 61 7C E4 AB 02	. 2 9 . 1 0 0 • d a t a | • • •
000000208	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000224	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000240	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000256	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000272	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000288	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000304	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000320	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000336	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000352	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000368	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000384	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000400	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000416	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000432	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000448	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000464	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000480	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000496	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000512	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •
000000528	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	• • • • • • • • • • • • • • • •

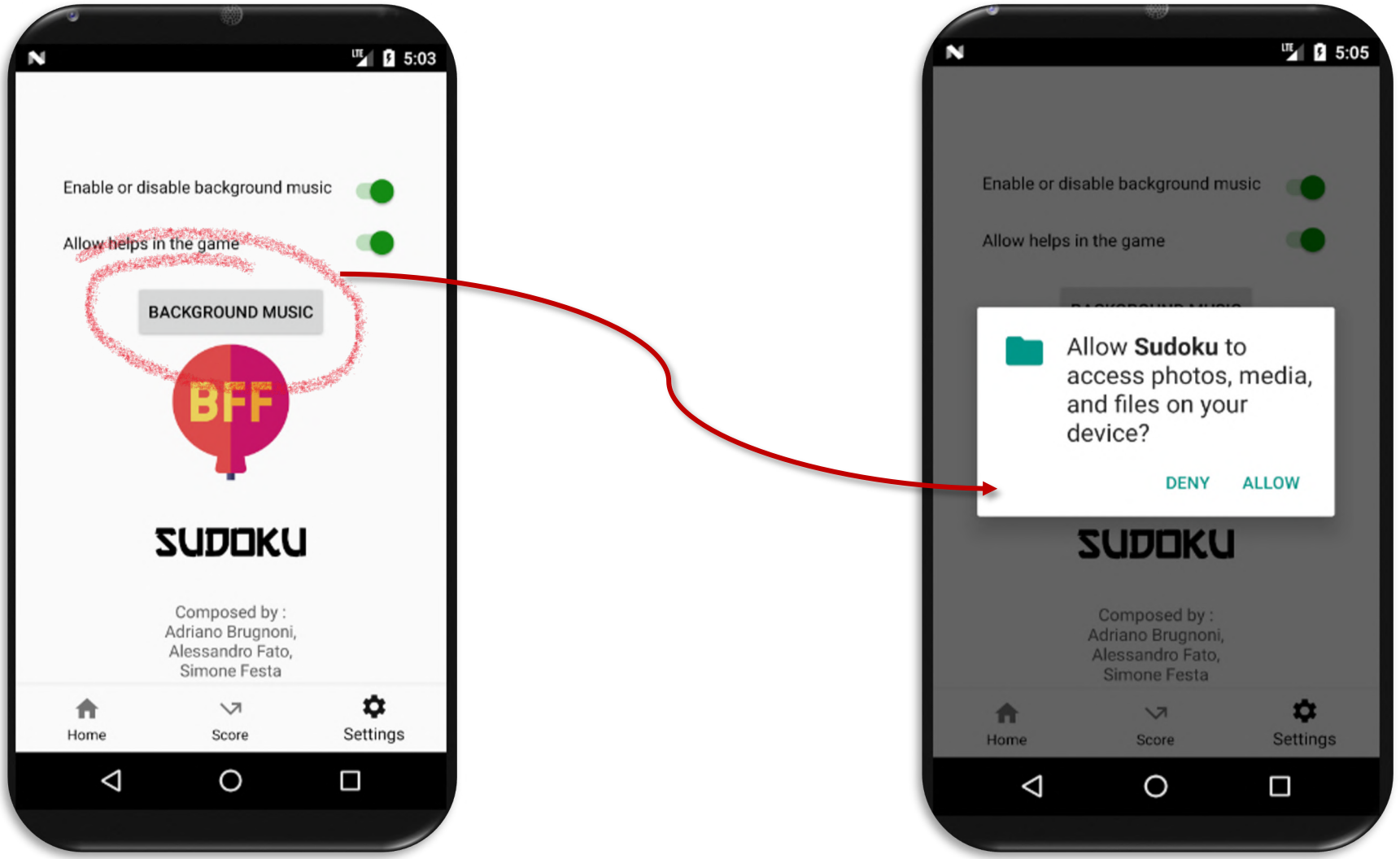
Classe WavFile – controllo header(1)

Address (Dec)	Hexadecimal (1 Byte)																Text (ASCII)															
000000000	52	49	46	46	44	E5	AB	02	57	41	56	45	66	6D	74	20	R	I	F	F	D	∘	∘	∘	W	A	V	E	f	m	t	
000000016	10	00	00	00	01	00	02	00	44	AC	00	00	10	B1	02	00	∘	∘	∘	∘	∘	∘	∘	∘	D	∘	∘	∘	∘	∘	∘	∘
000000032	04	00	10	00	4C	49	53	54	9C	00	00	00	49	4E	46	4F	∘	∘	∘	∘	L	I	S	T	∘	∘	∘	∘	I	N	F	O
000000048	49	41	52	54	0D	00	00	00	50	61	75	6C	20	44	65	73	I	A	R	T	∘	∘	∘	∘	P	a	u	l		D	e	s
000000064	6D	6F	6E	64	00	00	49	43	52	44	09	00	00	00	32	30	m	o	n	d	∘	∘	I	C	R	D	∘	∘	∘	∘	2	0
000000080	31	30	30	35	31	39	00	00	49	4E	41	4D	1F	00	00	00	1	0	0	5	1	9	∘	∘	I	N	A	M	∘	∘	∘	∘
000000096	54	68	65	20	54	68	65	6D	65	20	46	72	6F	6D	20	22	T	h	e		T	h	e	m	e		F	r	o	m		"
000000112	42	6C	61	63	6B	20	4F	72	70	68	65	75	73	22	00	00	B	l	a	c	k		O	r	p	h	e	u	s	"	∘	∘
000000128	49	50	52	44	2A	00	00	00	54	68	65	20	43	6F	6D	70	I	P	R	D	*	∘	∘	∘	T	h	e		C	o	m	p
000000144	6C	65	74	65	20	41	6C	62	75	6D	73	20	43	6F	6C	6C	l	e	t	e		A	l	b	u	m	s		C	o	l	l
000000160	65	63	74	69	6F	6E	3A	20	31	39	35	33	2D	31	39	36	e	c	t	i	o	n	:		1	9	5	3	-	1	9	6
000000176	33	00	49	53	46	54	0E	00	00	00	4C	61	76	66	35	38	3	∘	I	S	F	T	∘	∘	∘	∘	L	a	v	f	5	8
000000192	2E	32	39	2E	31	30	30	00	64	61	74	61	7C	E4	AB	02	.	2	9	.	1	0	0	∘	d	a	t	a	|	∘	∘	∘
000000208	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000224	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000256	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000272	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000288	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000304	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000320	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000336	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000352	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000368	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000384	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000400	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000416	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000432	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000448	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000464	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000480	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000496	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000512	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘
000000528	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘	∘

Classe WavFile – estrazione dati(2)

```
private void getWavFileInformations(byte[] buffer)
{
    //Dimensione
    WAV_FILE_DIMENSION = ((buffer[7] & 0xFF) << 24) | ((buffer[6] & 0xFF) << 16) | ((buffer[5] & 0xFF) << 8) | (buffer[4] & 0xFF) + 8;

    //Verifico metodo campionatura
    if((((buffer[21] & 0xFF) << 8) | ((buffer[20] & 0xFF)))==1)
        PCM = true;
    //Numero Canali
    if((((buffer[23] & 0xFF) << 8) | ((buffer[22] & 0xFF)))==1)
        MONO = true;
    else
        MULTICHANNEL = true;

    //Prelevo frequenza di campionamento
    SAMPLERATE=((buffer[27] & 0xFF) << 24) | ((buffer[26] & 0xFF) << 16) | ((buffer[25] & 0xFF) << 8) | (buffer[24] & 0xFF);
    if(SAMPLERATE == 0)
        SAMPLERATE=DEFAUL_SAMPLERATE;

    //Prelevo byte rate
    BYTE_RATE = ((buffer[31] & 0xFF) << 24) | ((buffer[30] & 0xFF) << 16) | ((buffer[29] & 0xFF) << 8) | (buffer[28] & 0xFF);

}
```

Change Background Music



```java
@Override
public void onClick(View v)
{
    String[] allPermissions = {Manifest.permission.READ_EXTERNAL_STORAGE};
    if(PermissionManager.askPermission(allPermissions, activity, PermissionManager.REQUEST_CODE_PERMISSION_READ_EXTERNAL_STORAGE)) {
        activity.launchActivity(PermissionManager.REQUEST_CODE_PERMISSION_READ_EXTERNAL_STORAGE);
    }
}
```

```java
public static boolean askPermission(String[] allPermissions, Activity activity, int requestCode)
{

    List<String> neededPermissions = new ArrayList<>();

    for(String s: allPermissions) {
        if(ContextCompat.checkSelfPermission(activity.getBaseContext(), s) != PackageManager.PERMISSION_GRANTED) {
            neededPermissions.add(s);
        }
    }

    if(!neededPermissions.isEmpty()) {
        ActivityCompat.requestPermissions(
                activity,
                neededPermissions.toArray(new String[neededPermissions.size()]),
                requestCode);
        return false;
    }

    return true;
}
```

```java
@Override
public void onClick(View v)
{
    String[] allPermissions = {Manifest.permission.READ_EXTERNAL_STORAGE};
    if(PermissionManager.askPermission(allPermissions, activity, PermissionManager.REQUEST_CODE_PERMISSION_READ_EXTERNAL_STORAGE)) {
        activity.launchActivity(PermissionManager.REQUEST_CODE_PERMISSION_READ_EXTERNAL_STORAGE);
    }
}
```

```java
public void launchActivity(int requestCode) {

    if (requestCode == PermissionManager.REQUEST_CODE_PERMISSION_READ_EXTERNAL_STORAGE) {
        Intent intent = new Intent(Intent.ACTION_PICK, MediaStore.Audio.Media.EXTERNAL_CONTENT_URI);
        if (intent.resolveActivity(getPackageManager()) != null) {
            this.startActivityForResult(intent , MenuActivity.GET_FILE_CODE);
        } else {
            Toast.makeText(getBaseContext(), getResources().getString(R.string.toast_directory_error), Toast.LENGTH_SHORT);
        }
    }
}
```

```java
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {

    if(PermissionManager.isAllGranted(grantResults)) {
        if(requestCode == PermissionManager.REQUEST_CODE_PERMISSION_READ_EXTERNAL_STORAGE) {
            launchActivity(PermissionManager.REQUEST_CODE_PERMISSION_READ_EXTERNAL_STORAGE);
        }
    }
    else {
        Toast.makeText(getBaseContext(), getResources().getString(R.string.toast_permission_denied), Toast.LENGTH_SHORT);
    }
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
}
```

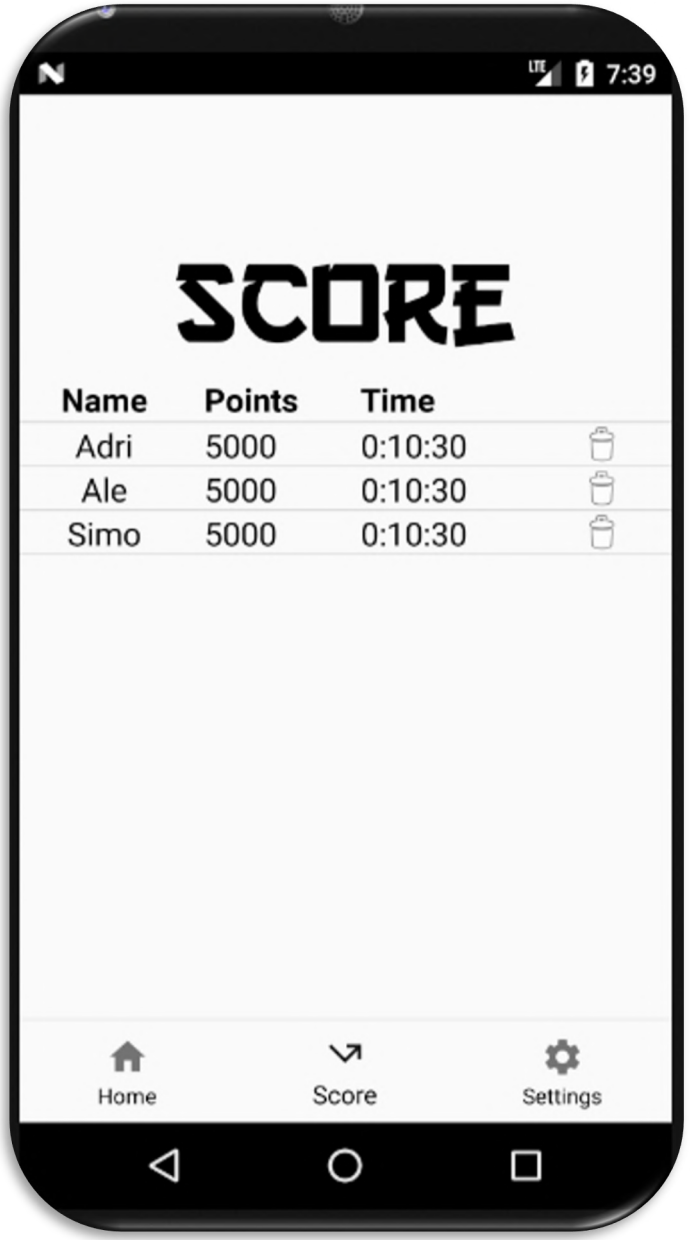
Classifica

Per rendere l'app ancora più interessante è stata implementata una classifica locale.

Caratteristiche:

- Memorizzazione: Giocatore, punti e tempo.

- Cancellazione.



Classifica - DB

Il database è implementato attraverso Room.

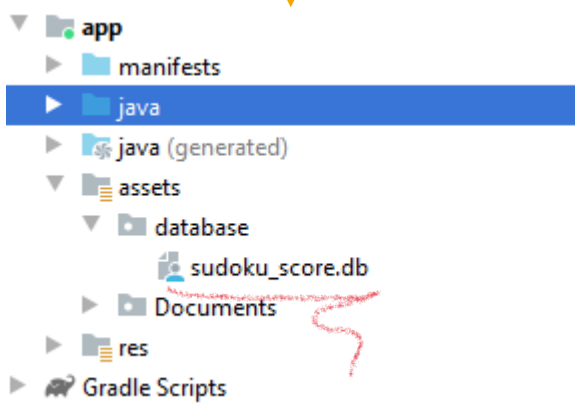
Ogni nuovo giocatore potrà sfidare noi sviluppatori trovando i nostri risultati nella classifica pre-installati.

```
private AppSudokuDatabase getDB() {
    return Room.databaseBuilder(activity.getApplicationContext(), AppSudokuDatabase.class,
"sudoku_score.db")
.allowMainThreadQueries()
.createFromAsset("database/sudoku_score.db")
.build();
}
```

Creazione DB - Zoom

Crea il database attuale a partire da uno precaricato nella directory interna:

«/data/data/it.bff.sudoku/databases/»




```
private AppSudokuDatabase
getDB() {
    return
Room.databaseBuilder(activ
ity.getApplicationContext(
),
AppSudokuDatabase.class,
"sudoku_score.db")
.allowMainThreadQueries()
.createFromAsset("database
/sudoku_score.db")
.build();
}
```

SudokuDAO

```
@Dao
public interface SudokuDAO {

  @Query("SELECT * FROM SudokuScore ORDER BY points
DESC")
  List<SudokuScore> getAll();

  @Insert(onConflict = OnConflictStrategy.REPLACE)
  void insertAll(SudokuScore... sudoku);

  @Delete
  void delete(SudokuScore sudoku);
}
```

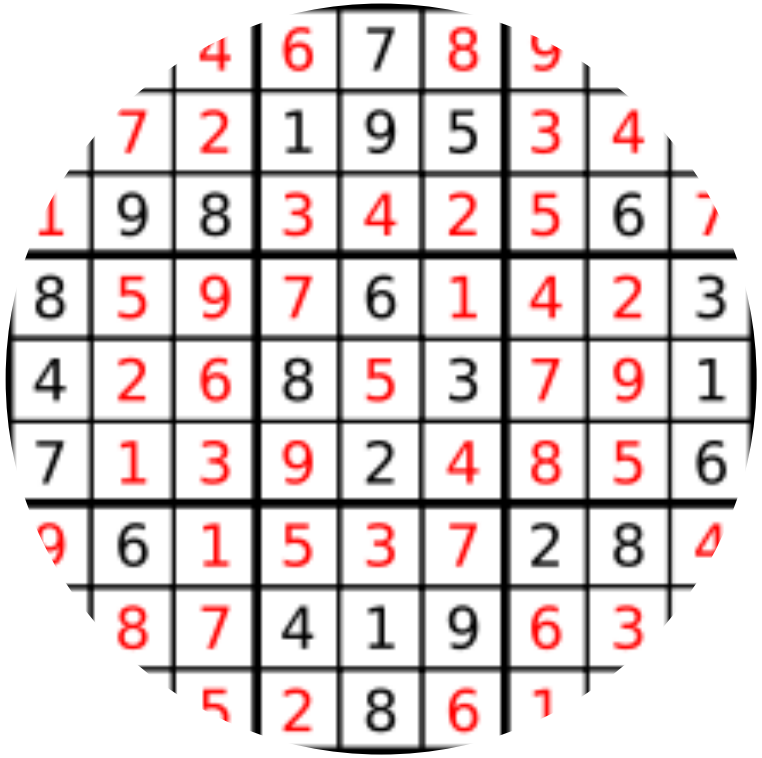
SudokuScore

```
public class SudokuScore
{
  @PrimaryKey(autoGenerate = true)
  @ColumnInfo(name="_id")
  public int idGame;
  @ColumnInfo(name="player")
  public String playerName;
  @ColumnInfo(name="points")
  String points;
  @ColumnInfo(name="timer")
  String timer;

  public void setPlayerName(String playerName) { this.playerName = playerName;}
  public void setPoints(String points) {this.points = points;}
  public void setTimer(String timer) {this.timer = timer;}

  public String getPlayerName() {return playerName;}
  public String getPoints() {return points}
  public String getTimer() {return timer;}
}
```

SudokuDAO - ZOOM

- Gli attributi da registrare riguardano nome utente (immesso al momento della vittoria), punteggio e timer, con un id primary key.
- Al momento della vittoria, mediante metodi set vengono memorizzati tali informazioni mediante query «insertAll», mentre con la query «delete» rimuoviamo la tupla.

```
@Dao
public interface SudokuDAO {

  @Query("SELECT * FROM SudokuScore ORDER BY
points DESC")
  List<SudokuScore> getAll();

  @Insert(onConflict =
OnConflictStrategy.REPLACE)
  void insertAll(SudokuScore... sudoku);

  @Delete
  void delete(SudokuScore sudoku);
}
```



Realizzato da

Adriano Brugnoni
Alessandro Fato
Simone Festa

Gruppo **BFF**