



Sudoku

By JGL

Struttura

- Griglia 9x9 di celle, per un totale di 81 celle
- La griglia è suddivisa in:
 - 9 righe
 - 9 colonne
 - 9 blocchi 3x3 di celle
- Ogni cella può contenere un numero da 1 a 9

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Scopo del Gioco

- Riempire le caselle vuote con numeri da 1 a 9, in modo tale che in ogni riga, in ogni colonna e in ogni blocco siano presenti tutte le cifre da 1 a 9.
- La condizione è che nessuna riga, colonna o blocco presentino due volte lo stesso numero.

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

MainActivity

La MainActivity gestisce la schermata Home dell' applicazione.

Permette all'utente di:

- Iniziare una nuova partita
- Consultare le impostazioni di gioco, tramite il bottone sul menu opzioni (SettingsActivity)
- Consultare le sue statistiche di gioco, tramite la navigation bar in basso (ResultActivity)



New Game

- L'applicazione utilizza la libreria Volley per il download delle griglie di gioco.
- I livelli sono generati in maniera randomica, e vengono scaricati tramite le API messe a disposizione dal sito 'sugoku.herokuapp.com'.
- L'applicazione dovrà poter accedere ad Internet, e poter verificare lo stato della connessione, di conseguenza è necessario inserire i seguenti permessi normali nel file Manifest:

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

New Game

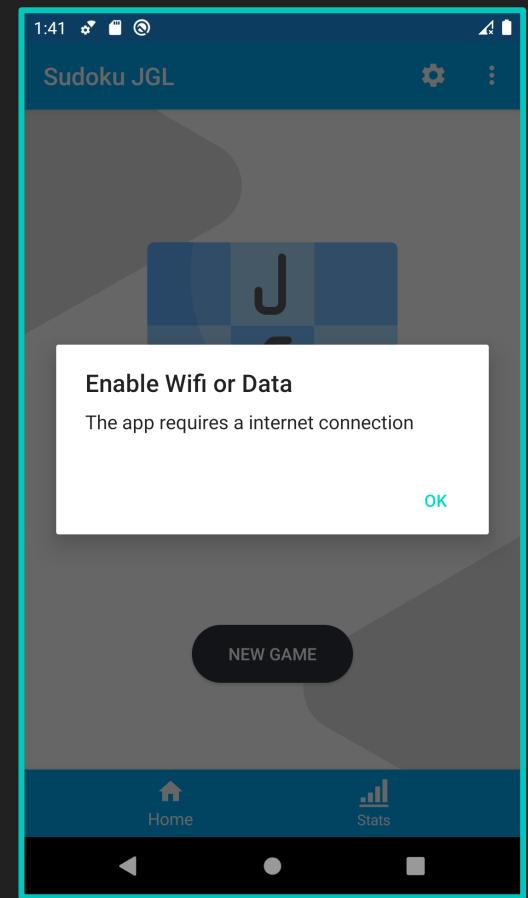
- Quando l'utente richiede di cominciare una nuova partita, tramite il metodo checkInternetConnection, viene effettuato un controllo della connessione Internet

```
private boolean checkInternetConnection() {  
    ConnectivityManager connectivityManager = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);  
    Network network = connectivityManager.getActiveNetwork();  
    NetworkCapabilities networkCapabilities = connectivityManager.getNetworkCapabilities(network);  
    return networkCapabilities != null && networkCapabilities.hasCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET);  
}
```

New Game

Sono due i casi possibili:

- Connessione presente: è possibile effettuare il download della griglia
- Connessione non presente: viene mostrato un Dialog che notifica la mancanza di connessione



New Game

- Accertata la presenza della connessione Internet, l'applicazione mostra un Dialog per scegliere il livello di difficoltà della partita, garantendo all'utente di misurarsi con livelli compatibili alla propria esperienza di gioco.



Difficulty

- A seconda del livello di difficoltà scelto, viene effettuato la corrispondente chiamata ad Internet per il download della griglia di gioco.

```
public void getSudokuEasy() {  
    difficulty = 1;  
    String url = "https://sugoku2.herokuapp.com/board?difficulty=easy";  
    apiCall(url);  
}  
  
public void getSudokuMedium() {  
    difficulty = 2;  
    String url = "https://sugoku2.herokuapp.com/board?difficulty=medium";  
    apiCall(url);  
}  
  
public void getSudokuHard() {  
    difficulty = 3;  
    String url = "https://sugoku2.herokuapp.com/board?difficulty=hard";  
    apiCall(url);  
}
```

Board

- La classe Volley presenta un metodo per convertire il JSONArray ricevuto in una lista di interi, che rappresentano i numeri della griglia di gioco scaricata.

```
private ArrayList<Integer> convert(JSONArray jsonArray) {  
    int [][] array = null;  
    ArrayList<Integer> al = new ArrayList<>();  
  
    int rows = 0, cols = 0;  
    try {  
        rows = jsonArray.length();  
        cols = 0;  
        for (int r = 0; r < rows; r++) {  
            JSONArray jsonRow = jsonArray.getJSONArray(r);  
            if (r == 0) {  
                cols = jsonRow.length();  
                array = new int[rows][cols];  
            }  
            for (int c = 0; c < cols; c++) {  
                array[r][c] = jsonRow.getInt(c);  
            }  
        }  
        catch (JSONException e) {  
            e.printStackTrace();  
        }  
  
        for (int r = 0; r < rows; r++) {  
            for (int c = 0; c < cols; c++) {  
                al.add(array[r][c]);  
            }  
        }  
    }  
    return al;  
}
```

Board

- La lista di numeri della griglia viene passata all'activity di gioco, insieme al livello di difficoltà selezionato.

```
JSONObject jsonObject = new JSONObject(response);
JSONArray jsonArray = jsonObject.getJSONArray(name: "board");
board = convert(jsonArray);

Intent intent = new Intent(activity, PlaySudokuActivity.class);
intent.putExtra(name: "board", board);
intent.putExtra(name: "difficulty", difficulty);
intent.putExtra(name: "new_game", value: true);
activity.startActivity(intent);
```

Game

L'applicazione implementa il vero e proprio gioco tramite le classi:

- Board, rappresenta la griglia di gioco
- Cell, rappresenta una singola cella della griglia
- SudokuGame, rappresenta la logica completa del gioco

Board

```
private int size;  
private List<Cell> cells;  
  
Board (int size, List<Cell> cells) {  
    this.size = size;  
    this.cells = cells;  
}
```

Cell

```
private boolean solve;
private int row;
private int col;
private int value;
private boolean valid;
private boolean isStartingCell;
private List<Integer> notes;

Cell(int row, int col, int value, boolean isStartingCell, List<Integer> notes, boolean solve) {
    this.row = row;
    this.col = col;
    this.value = value;
    this.isStartingCell = isStartingCell;
    this.notes = notes;
    this.solve = solve;

    if (isStartingCell) {
        valid = (value != 0);
    }
}
```

SudokuGame

- La classe SudokuGame converte la lista di numeri del gioco in una Board, caratterizzata da più Cell con una precisa posizione ed un eventuale valore assegnato.
- La logica viene gestita completamente dalla classe SudokuGame, la classe SudokuBoardView ha l'unico compito di disegnare la Board e di fare da tramite tra la Board e le azioni dell'utente.
- Viene inoltre costruita anche la Board corrispondente alla soluzione, per poter utilizzare le funzioni di soluzione automatica e di mossa intelligente.
- Sono, inoltre, disponibili due costruttori:
 - ❖ Uno per una nuova partita
 - ❖ Uno per la partita in corso

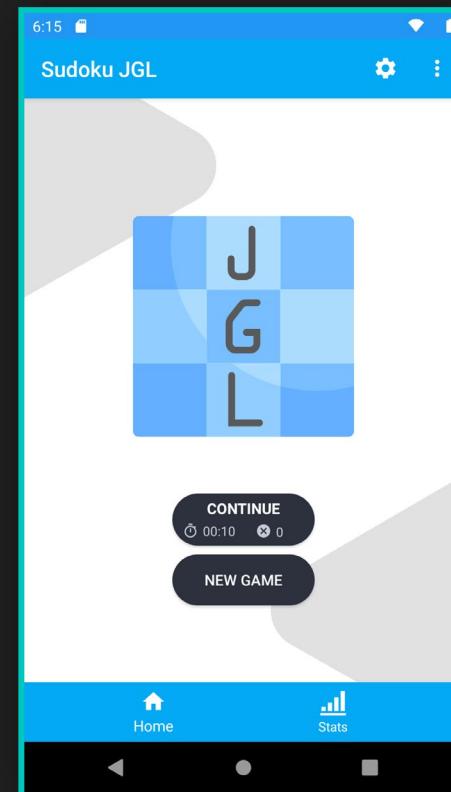
Nuova partita

```
List<Cell> cells = new ArrayList<>();
List<Cell> solvedCells = new ArrayList<>();
for (int i = 0; i < sizeBoard; i++) {
    Cell cell = new Cell( row: i / 9, col: i % 9, boardNumbers.get(i), isStartingCell: true, new ArrayList<Integer>(), solve: false);
    if (cell.getValue() == 0) cell.setStartingCell(false);
    cells.add(cell);

    // CELLS for solved board
    Cell solvedCell = new Cell( row: i / 9, col: i % 9, boardNumbers.get(i), isStartingCell: true, new ArrayList<Integer>(), solve: true);
    if (solvedCell.getValue() == 0) solvedCell.setStartingCell(false);
    solvedCells.add(solvedCell);
}
board = new Board( size: 9, cells);
```

Partita in corso

- È possibile riprendere una partita lasciata in sospeso
- Tale partita viene salvata su un database apposito quando si torna alla home o quando l'app viene chiusa
- Nel caso in cui sia presente la partita nel database, nella schermata di home viene reso visibile il pulsante CONTINUA con relative informazioni riguardo:
 - ❖ Tempo trascorso
 - ❖ Errori commessi



PlaySudokuActivity

- Per permettere alla Activity di effettuare operazioni sulla SudokuBoardView, sono stati implementati dei MutableLiveData per i dati più importanti (cella selezionate, celle della griglia, numero di errori, ecc.)
- Ognuno di essi implementa un proprio Observer dichiarato con un metodo apposito
- Quando un Observer "osserva" un cambiamento nei dati, esegue il relativo metodo

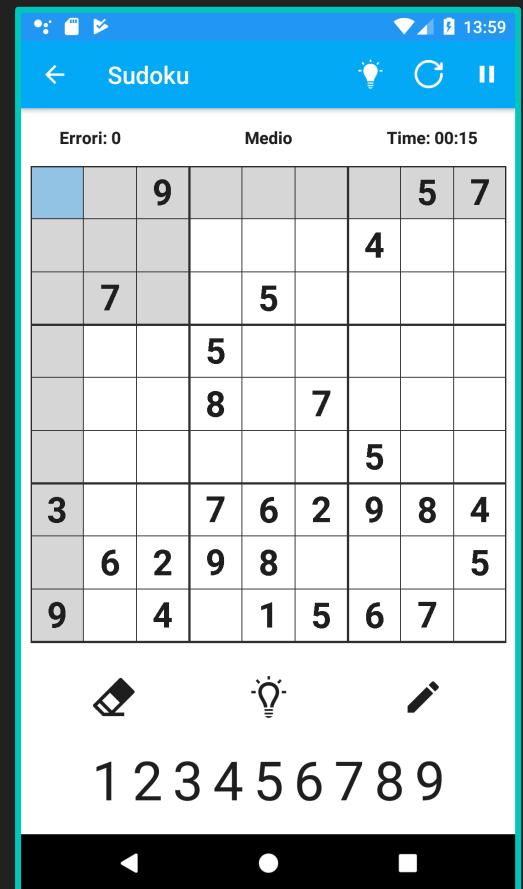
```
private class SetObserver {  
  
    private Observer<Cell> cellObserver;  
    private Observer<List<Cell>> cellsObserver;  
    private Observer<Boolean> isTakingNotesObserver;  
    private Observer<List<Integer>> notesObserver;  
    private Observer<Boolean> isInGameObserver;  
    private Observer<Integer> numErrObserver;
```

In Game

L'interfaccia grafica permette all'utente di compiere varie azioni sulla griglia di gioco:

Selezionando una cella è possibile:

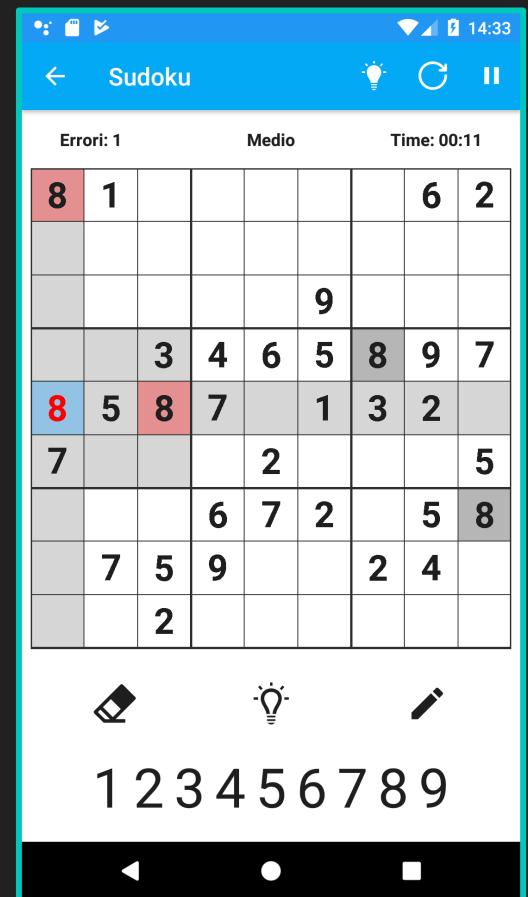
- Osservare la corrispondente riga, colonna e blocco messi in evidenza dall'app
- Inserire un numero dalla griglia in basso
- Utilizzare la gomma per cancellare un numero immesso
- Utilizzare la matita per annotare i numeri che, secondo l'utente, potrebbero occupare una cella



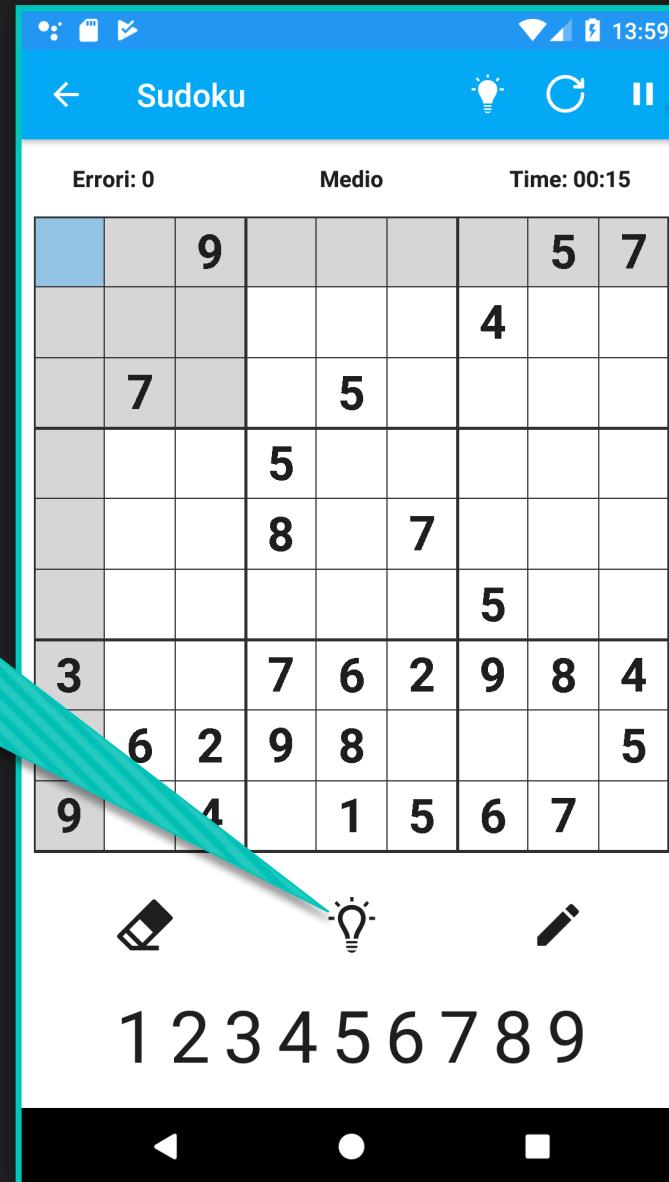
In Game

Vengono inoltre mostrati all'utente:

- Il numero di errori commessi
 - ❖ Aggiornato quando viene violata la regola di gioco. Sono inoltre evidenziati i numeri in conflitto con quello inserito.
- Il livello di difficoltà della partita
- Il tempo trascorso dall'inizio della partita
 - ❖ Bloccato quando l'app va in pausa



Permette all'utente di richiedere un aiuto all'applicazione, che inserisce in maniera randomica un numero corretto della soluzione

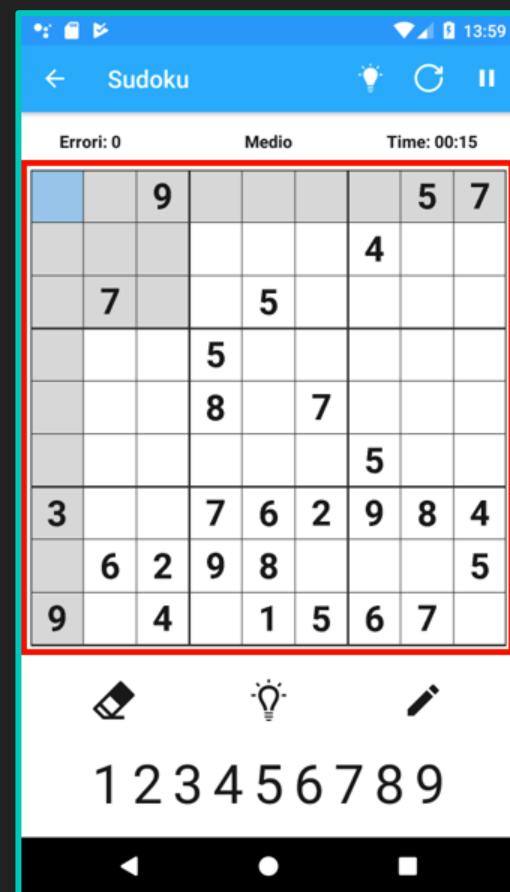


I comandi sulla barra consentono di :

- ❖ **Richiedere la risoluzione automatica della griglia**
- ❖ **Ripristinare il sudoku allo stato iniziale**
- ❖ **Mettere in pausa la partita**

SudokuBoardView

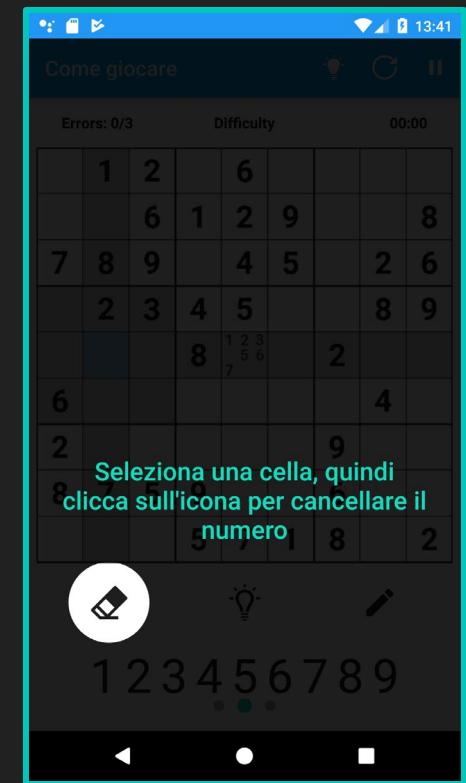
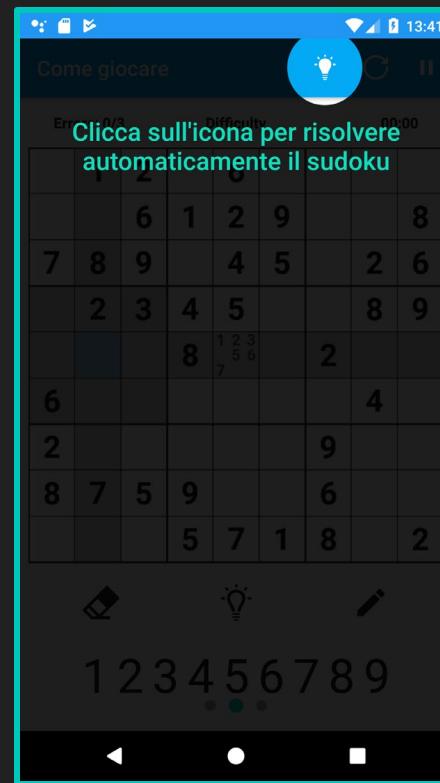
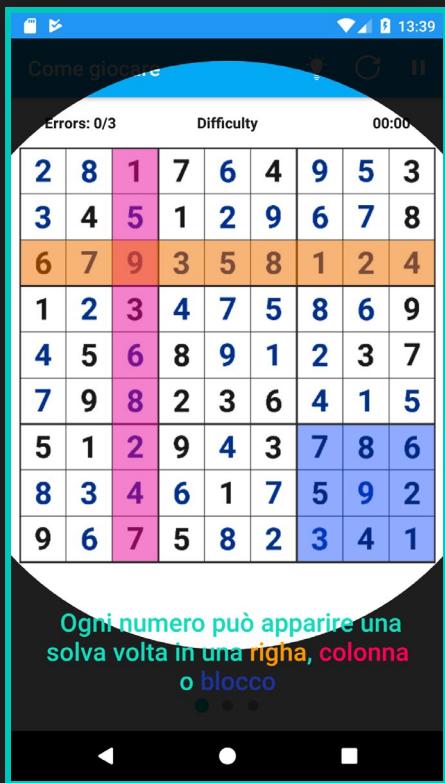
- La classe SudokuBoardView si occupa di disegnare la Board sull'activity di gioco, con cui va a interfacciarsi l'utente.
- Comprende un insieme di Paint necessari per disegnare le linee della griglia e i numeri all'interno delle celle
- La grandezza della griglia viene misurata dall'applicazione, in modo da adattarsi ad ogni display



Tutorial

- All'utente viene messo a disposizione un tutorial, mostrato automaticamente al primo avvio dell'applicazione.
- Inoltre, si ha la possibilità di visualizzare nuovamente le istruzioni, tramite la voce 'Come giocare' nel menu opzioni della MainActivity.
- Sostanzialmente viene spiegato il gioco e tutto ciò che compone l'interfaccia.

Tutorial



Settings

- Le impostazioni permettono all'utente di modificare le funzionalità e il comportamento dell'applicazione.
- La libreria Preference offre il modo migliore per costruire un'interfaccia finalizzata a mostrare all'utente ciò che può configurare.
- Un oggetto Preference rappresenta il blocco di base della libreria, e descrive una precisa impostazione.
- Un' activity per le impostazioni, di conseguenza, sarà caratterizzata da una gerarchia di sottoclassi di Preference, costruibile come una risorsa XML, che deve:
 - ❖ Avere come root tag <PreferenceScreen>
 - ❖ Essere posizionata nel package res/xml

Preferences

La risorsa XML restituisce una View associata ad una SharedPreferences, necessaria per salvare e recuperare i dati delle singole impostazioni.

Per questo motivo, un oggetto Preference è caratterizzato da:

- Key: chiave univoca per manipolare l'impostazione nella SharedPreferences
- DefaultValue: valore di default salvato nella SharedPreferences
- Icon: immagine associata all'impostazione
- Summary: descrizione dell'impostazione
- Title: titolo dell'impostazione

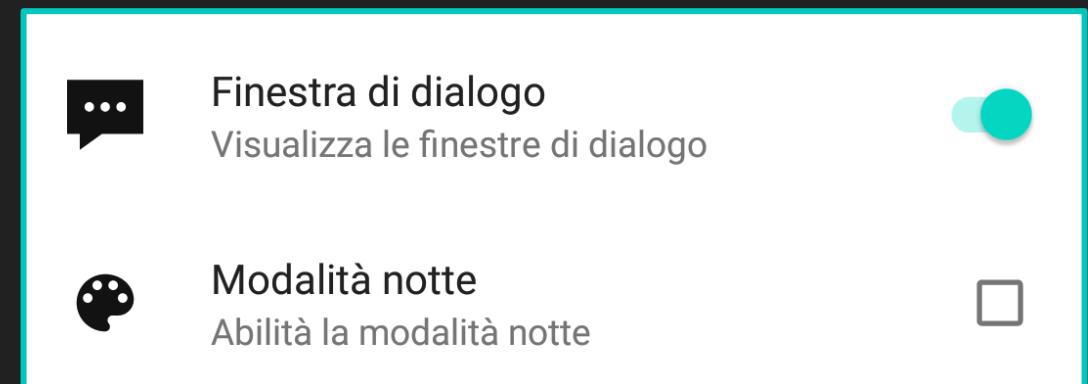
```
<SwitchPreferenceCompat  
    app:defaultValue="true"  
    app:icon="@drawable/icon_dialog"  
    app:key="dialog"  
    app:summary="Show dialog"  
    app:title="Dialog" />  
  
<CheckBoxPreference  
    app:defaultValue="false"  
    app:icon="@drawable/icon_night"  
    app:key="night_mode"  
    app:summary="Turn on night mode"  
    app:title="Night mode" />
```

PreferenceView

Ogni impostazione viene visualizzata come una riga di una ListView. All'interno dell'applicazione si fa uso di due sottoclassi di Preference, che hanno lo scopo di abilitare e disabilitare un'impostazione.

La differenza tra i due tipi sta nella View utilizzata per manipolare la preferenza:

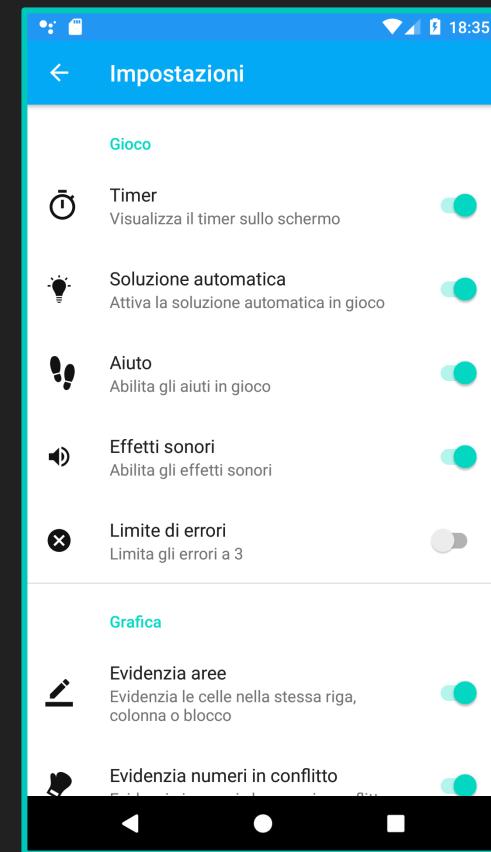
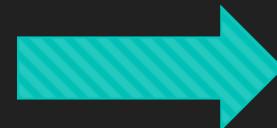
- SwitchPreferenceCompat, utilizza uno switch
- CheckBoxPreference, utilizza un checkbox



PreferenceCategory

- Tramite il tag PreferenceCategory è possibile raggruppare oggetti Preference e fornire un titolo che identifichi il gruppo.
- Viene utilizzata per mettere insieme impostazioni afferenti allo stesso ambito trattato.

```
<PreferenceCategory  
    android:title="Game">  
  
    <SwitchPreferenceCompat  
        app:defaultValue="true"  
        app:icon="@drawable/icon_timer"  
        app:key="timer"  
        app:summary="Enable timer on screen"  
        app:title="Timer" />
```



GetPreferences

- Classe introdotta per la gestione delle impostazioni configurabili dall'utente
- Utilizza la SharedPreferences associata alla ListView per:
 - ❖ Controllare i valori delle singole impostazioni
 - ❖ Gestire eventuali cambiamenti dovuti a modifiche dell'utente sulle singole impostazioni

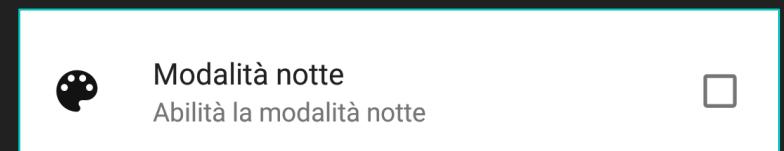
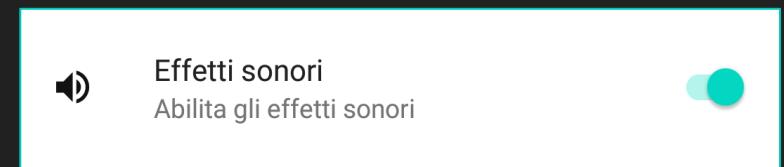
Check primo avvio

- Gestisce lo show del tutorial al primo avvio dell'applicazione:
 - ❖ Al primo avvio viene mostrato il tutorial e impostato il valore della chiave "first_launch" a false, in modo che esso sia d'ora in poi visibile solamente su richiesta esplicita dell'utente

```
public void checkFirstLaunch() {  
    if (preferences.getBoolean(key:"first_launch", defValue:true)) {  
        context.startActivity(new Intent(context, TutorialActivity.class));  
        preferences.edit().putBoolean("first_launch", false).apply();  
    }  
}
```

Check modifica impostazione

- Il metodo onSharedPreferenceChanged nella SettingsActivity viene invocato quando si verifica un cambiamento sui valori della SharedPreferences.
- Tramite i metodi della classe GetPreferences si effettua il controllo del valore della impostazione modificata.
- Sia per gli effetti sonori che per la modalità notte, è necessario effettuare la sostituzione del tema, seguita da una nuova creazione dell'Activity.



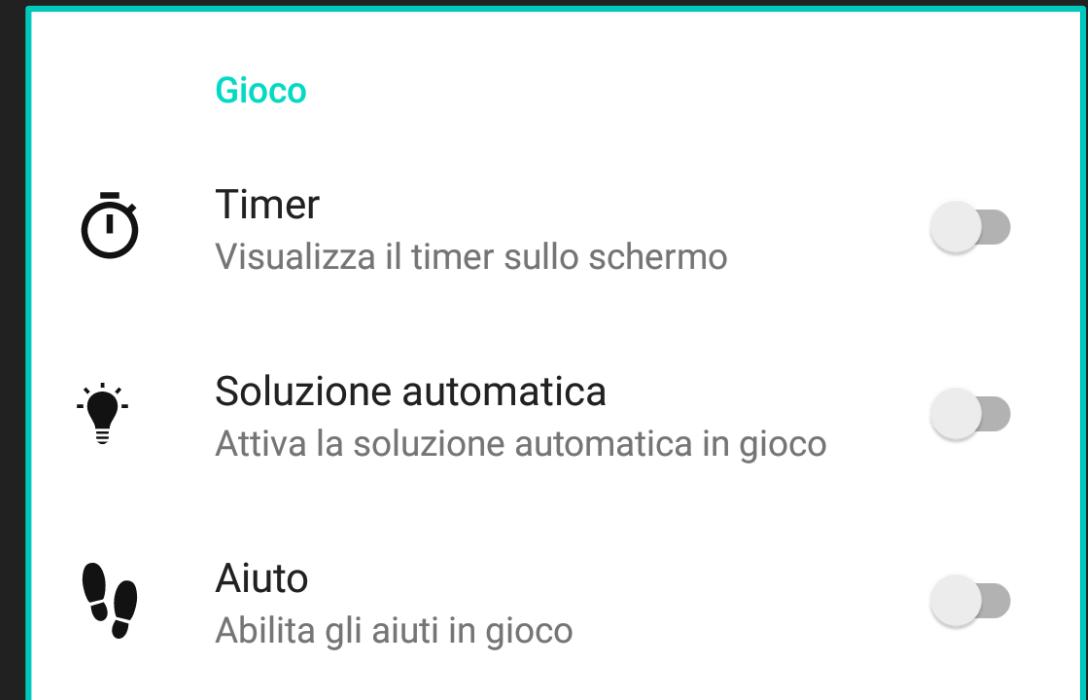
Metodo

```
@Override  
public void onSharedPreferenceChanged(SharedPreferences sharedpreferences, String key) {  
    if (key.equals("night_mode")) {  
        if (getPreferences.isNightModeOn()) {  
            AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES);  
            recreate();  
        }  
        else {  
            AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);  
            recreate();  
        }  
    }  
  
    else if (key.equals("sound_effects")) {  
        if (getPreferences.isSoundEffectsOn()) {  
            setTheme(R.style.AppTheme);  
            recreate();  
        }  
        else {  
            setTheme(R.style.DisabledSound);  
            recreate();  
        }  
    }  
}
```

```
public boolean isNightModeOn() { return preferences.getBoolean(key: "night_mode", defaultValue: false); }  
  
public boolean isSoundEffectsOn() { return preferences.getBoolean(key: "sound_effects", defaultValue: true); }
```

Impostazioni di gioco

- Alcune impostazioni permettono di alterare le funzionalità di gioco
- L'utente può abilitare/disabilitare:
 - ❖ Timer
 - ❖ Solver
 - ❖ Hint
 - ❖ Suoni
 - ❖ Limite di errori

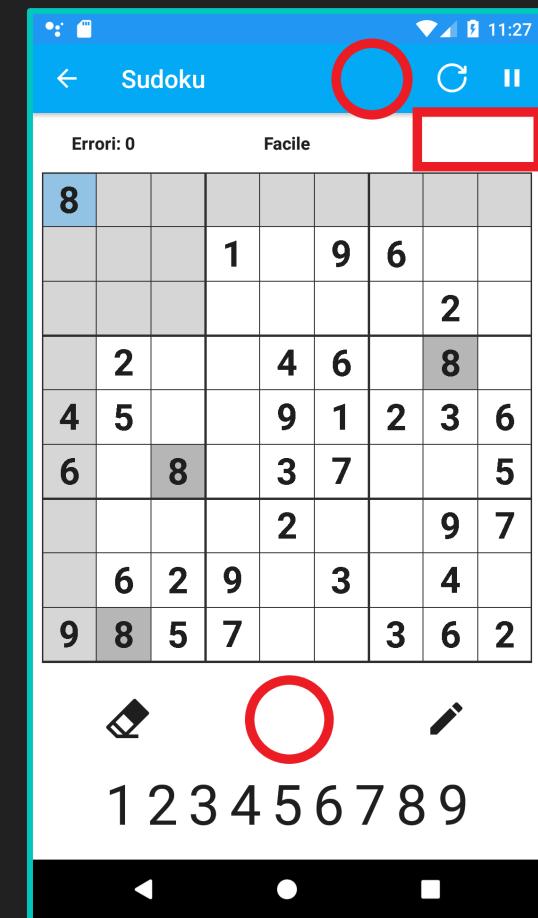


- La risoluzione automatica viene resa invisibile quando avviene la creazione del menu.

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    getMenuInflater().inflate(R.menu.menu_play, menu);  
    optionsMenu = menu;  
    if (!getPreferences.isSolverShow()) optionsMenu.findItem(R.id.mnuSolve).setVisible(false);  
    return true;  
}
```

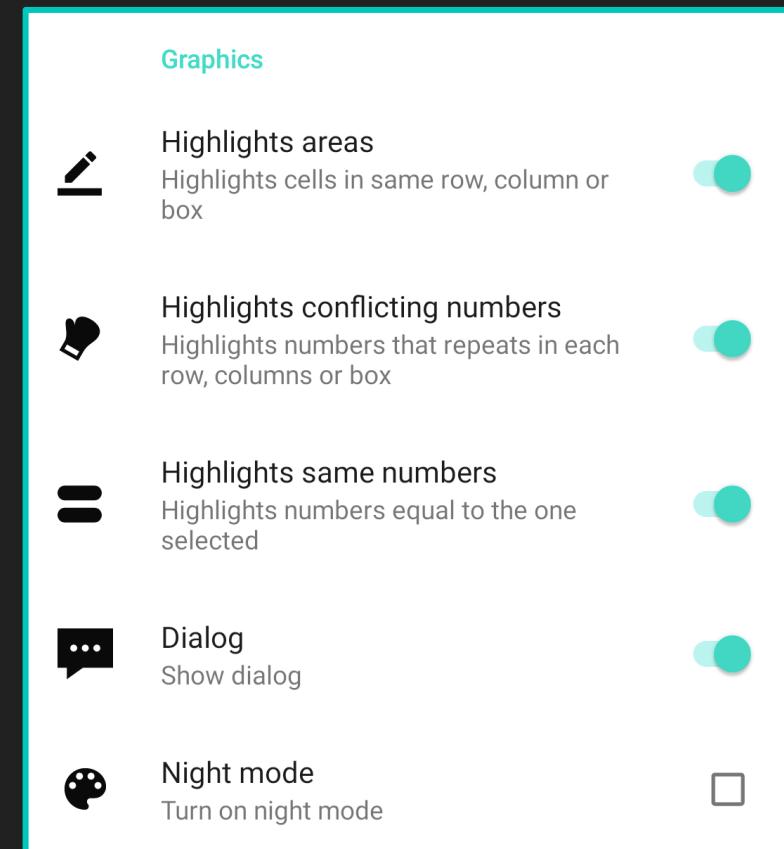
- Nell'Holder dell'Activity di gioco, vengono resi invisibili i suggerimenti e il timer. Si noti che il tempo viene comunque cronometrato per le statistiche di gioco.

```
btnHint = findViewById(R.id.btnHint);  
btnHint.setOnClickListener(this);  
if (!getPreferences.isHintShow()) btnHint.setVisibility(View.INVISIBLE);  
  
timer = findViewById(R.id.chronometer);  
timer.setFormat(MessageFormat.format(pattern: "{0} %s", "Time:"));  
if (newGame) timer.setBase(SystemClock.elapsedRealtime());  
else timer.setBase(SystemClock.elapsedRealtime() - time);  
if (!getPreferences.isTimerShow()) timer.setVisibility(View.INVISIBLE);  
timer.start();
```



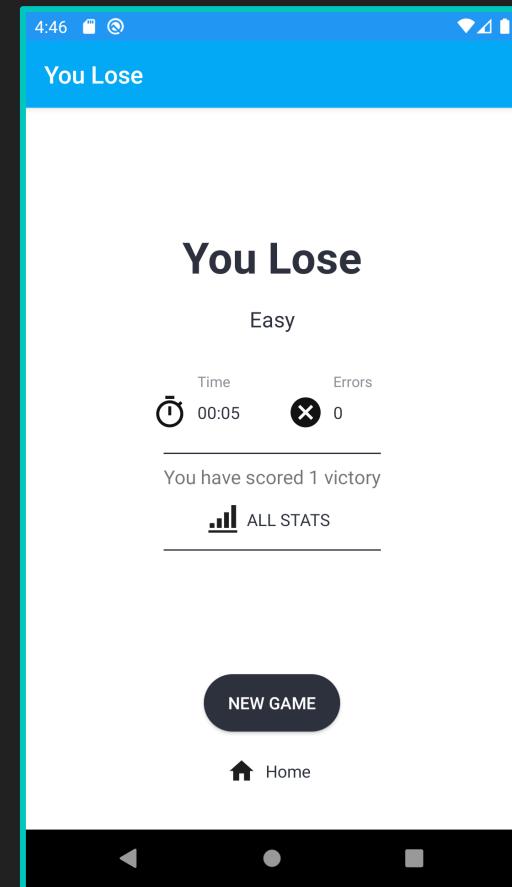
Impostazioni grafiche

- Alcune impostazioni permettono di alterare l'interfaccia di gioco
- L'utente può abilitare/disabilitare il display di:
 - ❖ Paint della SudokuBoardView
 - ❖ Night mode
 - ❖ Dialog



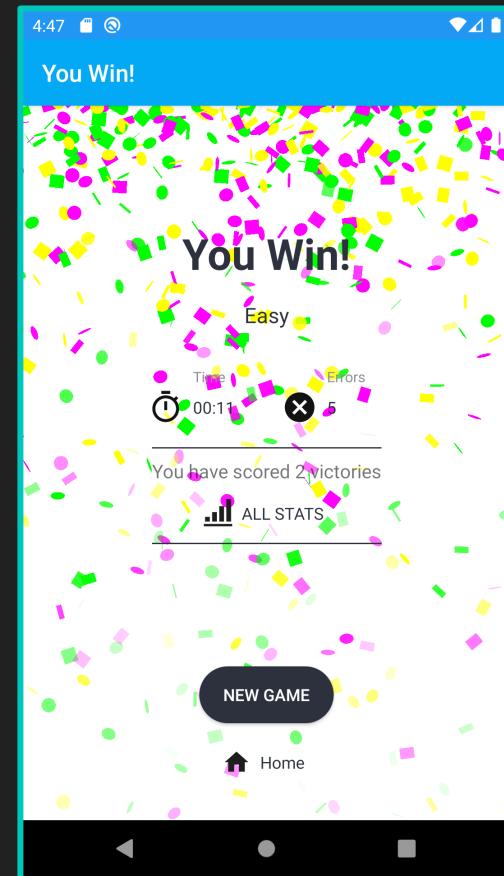
EndGameActivity

- Alla fine della partita, si ottiene una schermata finale accompagnata da un effetto sonoro a seconda dell'esito della partita
- È possibile, a questo punto:
 - ❖ Tornare alla home
 - ❖ Iniziare una nuova partita
 - ❖ Consultare le statistiche aggiornate
- In tale schermata vengono elencate le informazioni della partita appena completata:
 - ❖ Vittoria/Sconfitta
 - ❖ Tempo impiegato
 - ❖ Errori commessi



Vittoria

- Nel caso di vittoria, verrà impostata una KonfettiView
- Essa permette di mostrare a schermo dei coriandoli per festeggiare la vittoria dell'utente



Statistiche

L'applicazione utilizza la libreria Room, ed è caratterizzata da 2 database:

- AppDatabaseResult, memorizza le statistiche utente nella tabella Result
- AppDatabaseGame, memorizza i dati relativi all'ultima partita in corso nella tabella Game

```
@Entity
public class Result {
    @PrimaryKey(autoGenerate = true)
    public int resultId;
    @ColumnInfo(name="boolWin")
    private boolean boolWin;
    @ColumnInfo(name="intType")
    private int intType;
    @ColumnInfo(name="longTime")
    private long longTime;
    @ColumnInfo(name="intErr")
    private int intErr;
```

```
@Entity
public class Game {
    @PrimaryKey(autoGenerate = true)
    public int id;
    @ColumnInfo(name="listCells")
    private List<Cell> listCells;
    @ColumnInfo(name="listCellsSolved")
    private List<Cell> listCellsSolved;
    @ColumnInfo(name="intDifficulty")
    private int intDifficulty;
    @ColumnInfo(name="intErrors")
    private int intErrors;
    @ColumnInfo(name="longTime")
    private long longTime;
    @ColumnInfo(name="boolErrLimiter")
    private boolean errLimiter;
```

Game database

AppDatabaseGame fa uso di una classe DataConverter che si occupa di convertire i dati dell'entità Game.

In modo più dettagliato la classe DataConverter comprende due metodi:

- Uno per convertire la lista di celle della partita in una stringa da salvare nel DB
- L'altro per convertire la stringa nel DB nella lista di celle della partita da riprendere

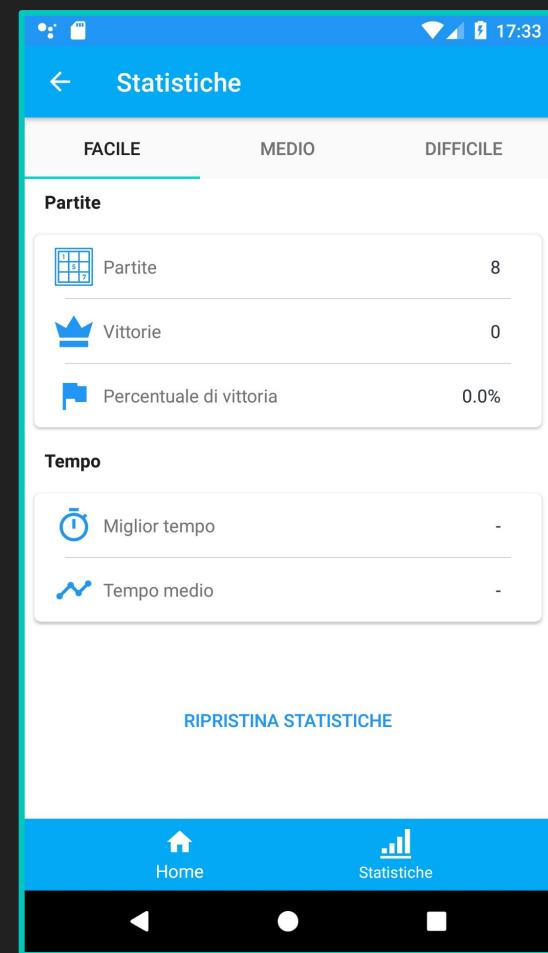
```
@TypeConverter  
public List<Cell> toListCell(String cellsString) {  
    if (cellsString == null) return null;  
    Gson gson = new Gson();  
    Type type = new TypeToken<List<Cell>>() {}.getType();  
    return gson.fromJson(cellsString, type);  
}
```

```
@TypeConverter  
public String fromListCell(List<Cell> cells) {  
    if (cells == null) return null;  
    Gson gson = new Gson();  
    Type type = new TypeToken<List<Cell>>() {}.getType();  
    return gson.toJson(cells, type);  
}
```

Result Activity

Le statistiche di gioco mostrate all'utente sono:

- Numero di partite
- Numero di vittorie
- Percentuale di vittoria
- Miglior tempo
- Tempo medio



Result Activity

La ResultActivity è caratterizzata da:

- FragmentPagerAdapter: contenitore di Fragment
- TabLayout: layout orizzontale per mostrare dei tab
- ViewPager: mantiene ciò che deve essere mostrato per ogni tab

PagerAdapter

Il PagerAdapter consente di creare una lista di Fragment collegati tra loro.

Necessita di:

- Una lista di Fragment
- Relativi titoli

```
public class PagerAdapter extends FragmentPagerAdapter {

    private List<Fragment> fragmentList = new ArrayList<>();
    private List<String> fragmentTitleList = new ArrayList<>();

    public void addFragment(Fragment fragment, String title) {
        fragmentList.add(fragment);
        fragmentTitleList.add(title);
    }

    public PagerAdapter(FragmentManager fm) { super(fm); }

    @Nullable
    @Override
    public CharSequence getPageTitle(int position) { return fragmentTitleList.get(position); }

    @NonNull
    @Override
    public Fragment getItem(int position) { return fragmentList.get(position); }

    @Override
    public int getCount() { return fragmentList.size(); }
}
```

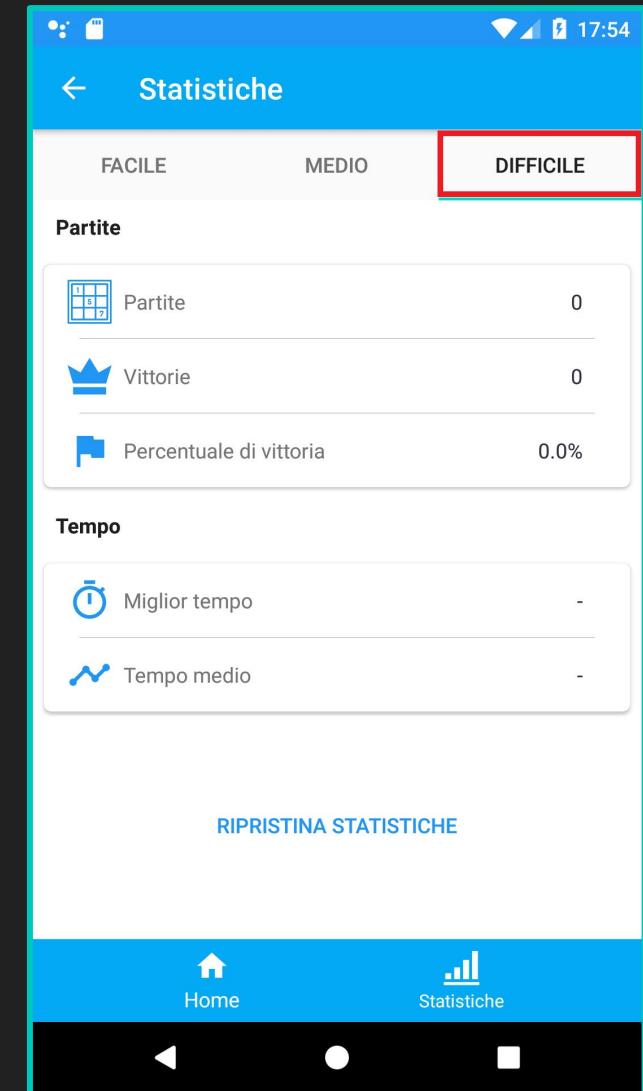
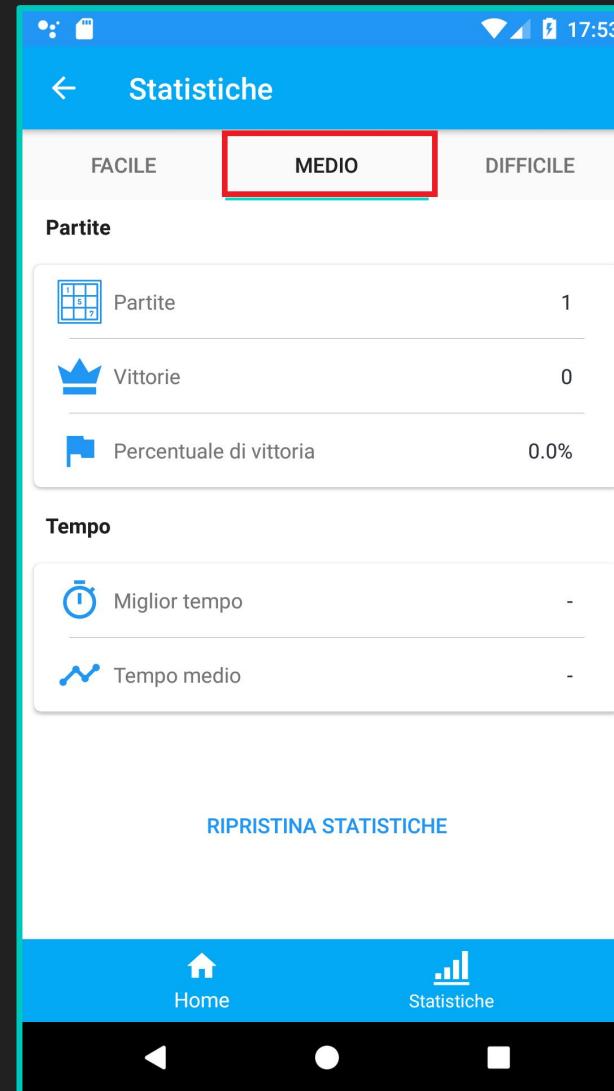
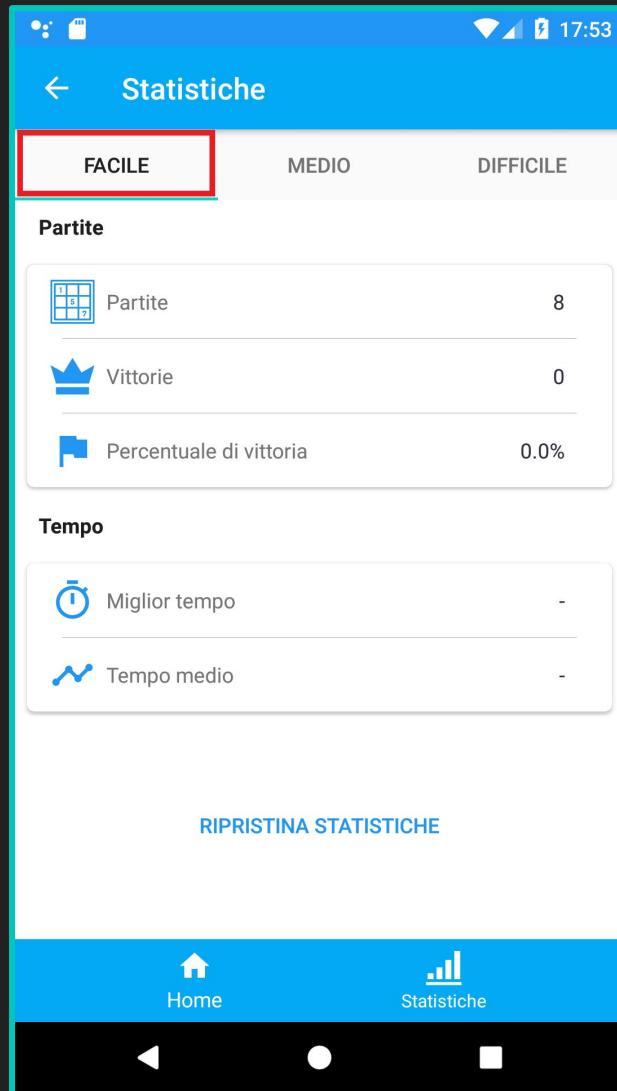
PagerAdapter

- Ogni Fragment mostra le statistiche di gioco relative ad un preciso livello di difficoltà.
- Il ViewPager utilizza il FragmentPagerAdapter, di conseguenza ad ogni tab viene associato un preciso Fragment.
- Il metodo onTabSelected permette di cambiare il Fragment del ViewPager in base alla posizione del tab selezionato.

```
private void setupViewPager(ViewPager viewPager) {  
    pagerAdapter = new PagerAdapter(getSupportFragmentManager());  
    pagerAdapter.addFragment(new FragmentEasy(), "Easy");  
    pagerAdapter.addFragment(new FragmentMedium(), "Medium");  
    pagerAdapter.addFragment(new FragmentHard(), "Hard");  
    viewPager.setAdapter(pagerAdapter);  
}
```

```
@Override  
public void onTabSelected(TabLayout.Tab tab) {  
    viewPager.setCurrentItem(tab.getPosition());  
}
```

Grazie a questa realizzazione, l'utente ha la possibilità di osservare le proprie statistiche per ogni livello di difficoltà.



Grazie dell' attenzione

Realizzato da:

- Jacopo Fabi
- Gabriele Quatrana
- Luca Santopadre