




# JGL Sudoku

**Analisi di progetto**

**Rev. 1.0**


*Jacopo Fabi, Gabriele Quatrana, Luca Santopadre*

Rev.	Data	Descrizione Modifica
1.0	15/06/2020	Prima stesura del documento


	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>2</b> di <b>46</b>	

# SOMMARIO


Sommario .....	2
1    Introduzione .....	5
1.1    Scopo.....	5
1.2    Riferimenti.....	5
2    Analisi dei requisiti e scelte effettuate.....	6
2.1    Analisi dei requisiti.....	6
2.2    Scelte di progetto.....	6
2.3    Librerie utilizzate.....	7
3    Manifest.....	8
3.1    Permessi .....	8
3.2    Orientamento.....	8
3.3    Activity.....	8
4    Activity .....	9
4.1    MainActivity .....	9
4.2    AboutActivity.....	10
4.3    EndGameActivity.....	10
4.4    PlaySudokuActivity.....	11
4.5    ResultActivity .....	13
4.6    SettingsActivity.....	14
4.7    TutorialActivity.....	15
5    Fragment .....	16

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>3</b> di <b>46</b>	

5.1	FragmentResult.....	16
5.2	FragmentTutorial .....	16
5.3	FragmentSettings .....	16
6	Adapter.....	17
7	Classi custom .....	18
7.1	GetPreferences .....	18
7.2	RestartApp .....	19
7.3	DataConverter.....	19
7.4	TimeParser .....	19
8	Dialog.....	20
9	ViewModel .....	21
10	Database.....	22
11	Volley.....	23
12	Game .....	24
12.1	Board.....	24
12.2	Cell.....	24
12.3	SudokuGame .....	25
13	View.....	28
14	Drawable .....	30
15	Colors.....	31
16	Strings.....	32
17	Layout .....	33
17.1	MainActivityLayout .....	33

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 4 di 46	

17.2	PlaySudokuActivityLayout.....	34
17.3	EndGameActivityLayout.....	35
17.4	AboutActivityLayout.....	36
17.5	BottomSheetLayout .....	37
17.6	TutorialActivityLayout .....	38
17.7	ResultActivityLayout.....	39
17.8	ResultFragmentLayout .....	40
17.9	TutorialFragmentLayout .....	41
17.10	PreferenceScreenLayout.....	42
17.11	FancyShowCaseViewLayout .....	42
18	Menu .....	43
18.1	MainActivityMenu.....	43
18.2	PlaySudokuActivityMenu .....	44
18.3	BottomNavigationMenu .....	45
19	Style & Locale .....	46

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 5 di 46	

# 1 Introduzione

## 1.1 Scopo

JGL Sudoku è il progetto realizzato per la prova pratica finale relativa al corso di Mobile Programming 2019-2020.

Il progetto intende sviluppare un'applicazione android del gioco del Sudoku che soddisfi i seguenti requisiti minimi:

- Un'interfaccia attraente ed intuitiva
- Un generatore di schemi automatico
- Un database per salvare i risultati di gioco
- Suggestere "mosse" all'utente
- Permettere appunti per ogni cella del sudoku
- Localizzazione

Ed altre funzionalità aggiuntive:

- Un tutorial di utilizzo per l'utente
- Mantenere progressi della partita in corso

## 1.2 Riferimenti

<https://it.wikipedia.org/wiki/Sudoku>

<https://www.wikihow.it/Risolvere-il-Sudoku>


<https://sugoku.herokuapp.com>

<https://developer.android.com/reference/android/preference/PreferenceScreen>

<https://github.com/faruktoptas/FancyShowCaseView>

<https://github.com/DanielMartinus/Konfetti>

<https://iconmonstr.com/>

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 6 di 46	

## 2 Analisi dei requisiti e scelte effettuate

### 2.1 Analisi dei requisiti

La griglia di gioco dovrà essere generata in modo casuale e permettere all'utente di selezionare il livello di difficoltà.

Durante il gioco l'utente avrà a disposizione un timer, la possibilità di inserire note per ogni cella, gli aiuti di gioco, la possibilità di mettere in pausa il gioco e di riprendere la partita successivamente.

Le impostazioni dovranno permettere all'utente di abilitare/disabilitare gli aiuti nel gioco, il timer di gioco, il numero di errori massimo.


Sarà necessario salvare e mostrare risultati e statistiche per l'utente.

L'utente dovrà essere in grado di salvare una partita e riprenderla anche una volta chiusa l'app.

L'utente potrà scegliere la modalità light o dark dell'interfaccia a suo gradimento.

### 2.2 Scelte di progetto


- Le griglie di gioco verranno scaricate da internet per mezzo di API messe a disposizione da <https://sugoku.herokuapp.com>. La gestione del download viene eseguita dalla libreria **Volley** che scarica in modo casuale le griglie in formato JSON, in base al livello di difficoltà selezionato. Verrà utilizzata anche la libreria Google **Gson** per la gestione più efficiente dei file JSON.
- Il salvataggio dei dati avverrà per mezzo della libreria **Room**: una tabella si occuperà di tenere traccia delle statistiche di gioco dell'utente mentre un'altra si occuperà di salvare l'ultima partita in corso.
- Verrà utilizzato un **Menu** nella Home dove verrà inserito un bottone di **Settings** che permette di accedere alle impostazioni dell'app, gestite dalla libreria **Preference** di Android, la quale mette a disposizione un menu di tipo **ListView** associata alle **SharedPreferences** per permettere alle varie activity di accedere alle impostazioni utente quali, ad esempio, disabilitare/abilitare il suono, gli aiuti, la night mode ecc.
- Un **Menu bottom** viene inserito nella parte inferiore dello schermo, e mostrerà 2 bottoni, uno per "torna alla Home" e uno per "accesso alle statistiche".
- Le statistiche di gioco verranno mostrate all'utente per mezzo di **Fragment**, divisi per livello di difficoltà (Easy, Medium, Hard).
- Il ciclo di vita del gioco verrà gestito dalla **ViewModel** per il salvataggio e ripristino dello stato di gioco.

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>7</b> di <b>46</b>	

- Dei **Dialog** verranno mostrati all'utente per l'interazione, come ad esempio la scelta del livello o la conferma di uscita dalla partita.
- Ogni Activity delegherà ad una sottoclasse **Holder** la gestione e la logica delle view.

## 2.3 Librerie utilizzate

Libreria	Ver.	Package
Room-runtime	2.2.5	androidx.room
Viewmodel	1.1.1	android.arch.lifecycle
Volley	1.1.1	com.android.volley
Gson	2.8.6	com.google.code.gson
Preference	1.1.1	androidx.preference
RecyclerView	1.1.0	androidx.recyclerview
AppCompat	1.1.0	androidx.appcompat
ConstraintLayout	1.1.3	androidx.constraintlayout
Material	1.1.0	com.google.android.material
Konfetti	1.2.0	nl.dionsegijn
FancyShowCaseView	1.3.0	me.topas.fancyshowcase

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 8 di 46	

## 3 Manifest

### 3.1 Permessi

L'app dovrà poter accedere ad Internet per scaricare le griglie di gioco, e poter verificare lo stato della connessione, quindi inseriamo i seguenti permessi normali:

```
<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" />
```

### 3.2 Orientamento

L'app viene fornita solo in modalità *Portrait* per una migliore giocabilità e visibilità della griglia.

### 3.3 Activity

Le seguenti activity vengono dichiarate nel manifest e dettagliate in seguito:

```
<activity
    android:name=".activity.HomeActivity"
    android:theme="@style/SplashTheme">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />


        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

<activity android:name=".activity.PlaySudokuActivity" />
<activity android:name=".activity.EndGameActivity" />
<activity android:name=".activity.ResultActivity" />
<activity android:name=".activity.SettingsActivity" />
<activity android:name=".activity.TutorialActivity" />
<activity android:name=".activity.AboutActivity" />
```



	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 9 di 46	

## 4 Activity

### 4.1 MainActivity

È l'activity di default e si occupa dell'avvio dell'applicazione e di gestire la schermata Home. Nel metodo *onCreate* andiamo ad inizializzare: una classe *GetPreference* di supporto alle *SharedPreferences*, il database dei risultati e della partita in corso, la classe *volley* per il download delle griglie di gioco, il menu impostazioni, il menu in basso per l'accesso alle statistiche e la sottoclasse *Holder* per la gestione delle view.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Get preferences and check first launch
    getPreferences = new GetPreferences(this);
    getPreferences.checkFirstLaunch();
    getPreferences.checkEnabledSound();
    setContentView(R.layout.activity_main);
    // Set title
    setTitle(getResources().getString(R.string.app_title));
    createDB();
    volleySudoku = new VolleySudoku(this);
    loadingDialog = new LoadingDialog(this);
    bottomDialog = new BottomDialog(this, volleySudoku, loadingDialog);
    bottomDialog.makeDialog();
    new Holder();
}
```

La sottoclasse *Holder* che permette di gestire le view nella schermata iniziale, quali il *btnPlay* per l'avvio di una nuova partita, *btnContinue* per continuare una partita esistente e la *navigation bar* in basso.


```
private Holder() {
    btnPlay = findViewById(R.id.btnPlay);
    btnPlay.setOnClickListener(this);

    nav = findViewById(R.id.bottom_navigation);
    nav.setOnNavigationItemSelectedListener(this);

    btnContinue = findViewById(R.id.btnContinue);
    btnContinue.setOnClickListener(this);
    if (dbGame.gameDAO().getGame() != null) {
        btnContinue.setVisibility(View.VISIBLE);
        tvTime = findViewById(R.id.tvTime);
        tvErrors = findViewById(R.id.tvErrors);

        String time = timeParser.parse(dbGame.gameDAO().getGame().getLongTime());
        tvTime.setText(time);

        tvErrors.setText(String.valueOf(dbGame.gameDAO().getGame().getIntErrors()));
    }
}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 10 di 46	

## 4.2 AboutActivity

L'activity viene chiamata al click sulla voce "About" del menu e mostra le INFO sulla versione dell'applicazione e l'API utilizzate per creare le griglie di gioco.

## 4.3 EndGameActivity

Si occupa della logica di fine gioco: nel metodo *onCreate* vengono consultati i dati della *Intent* creata dalla *PlaySudokuActivity* alla fine della partita. In particolare, vengono mostrati il tempo residuo, il livello, il numero di errori e l'esito della partita (vinto/perso). Sono due le schermate possibili mostrate a fine partita:

- L'utente vince: viene mostrata la schermata di vittoria.
- L'utente perde: viene mostrata la schermata di sconfitta.

`@Override`

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);


    // Get preferences and check sound effects
    getPreferences = new GetPreferences(this);
    getPreferences.checkEnabledSound();
    setContentView(R.layout.activity_end_game);

    // Set restart app and time parser
    restartApp = new RestartApp(this);
    timeParser = new TimeParser();

    // The end screen will contains all the game info (timer, errors, difficulty...)
    Intent intent = getIntent();
    endTime = intent.getLongExtra("timer", 0);
    difficulty = intent.getIntExtra("difficulty", 0);
    numErrors = intent.getIntExtra("error", 0);
    win = intent.getBooleanExtra("win", true);

    // Set win/lose sound
    setEndSound();
    volleySudoku = new VolleySudoku(this);
    loadingDialog = new LoadingDialog(this);
    MainActivity.loadingDialog = loadingDialog;
    bottomDialog = new BottomDialog(this, volleySudoku, loadingDialog);
    bottomDialog.makeDialog();
    new Holder();
}
```

Nel caso in cui l'utente vinca, verrà utilizzata la libreria *Konfetti* (<https://github.com/DanielMartinus/Konfetti>), che permette di creare una view per visualizzare a schermo dei coriandoli per festeggiare la vittoria dell'utente. Inoltre, viene impostato un suono diverso in base all'esito della partita.

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 11 di 46	

## 4.4 PlaySudokuActivity

È l'activity principale e si occupa della gestione della schermata di gioco. Viene avviata quando l'utente clicca su "gioca" e seleziona il livello. All'avvio, nel metodo *onCreate*, vengono presi i parametri "board" e "difficulty" dalla *Intent* creata dalla classe *VolleySudoku*, quindi viene creata la partita inizializzando il *PlaySudokuViewModel*.


```
// Set the new Game
viewModel = new PlaySudokuViewModel(board);
sudokuBoardView = findViewById(R.id.sdkView);
sudokuBoardView.registerListener(this);
```

I metodi *updateNoteTakingUI*, *updateHigLightedKeys*, *updateStateGame*, *updateNumErrors* sono utilizzati rispettivamente per aggiornare lo stato *TakingNotes*, evidenziare i numeri usati nelle note della cella, aggiornare lo stato della partita e aggiornare il numero di errori.

```
// If the state will change in TakingNote, the button icon color will change
private void updateNoteTakingUI(Boolean bool) {
    if (bool) {
        holder.btnNotes.setColorFilter(getColor(R.color.colorPrimaryDark));
    }
    else {
        holder.btnNotes.setColorFilter(getColor(R.color.mainColor));
    }
}
```

```
// Numbers used in notes will change in color
private void updateHigLightedKeys(List<Integer> list) {
    int index = 0;
    for (TextView btn : holder.buttons) {
        int color;
        if (list.contains(index + 1)) {
            color = getResources().getColor(R.color.colorPrimaryDark, null);
        }
        else {
            color = getColor(R.color.mainColor);
        }
        btn.setTextColor(color);
        index++;
    }
}
```

```
// The EndGame is specified in a LiveData
private void updateStateGame(Boolean inGame) {
    if (!inGame) {
        setEndGame();
    }
}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 12 di 46	

```
// Every time the player will make a mistake, the number of errors will be checked
private void updateNumErrors(Integer numErrors) {
    numErr = numErrors;
    if (getPreferences.isErrorLimiter())
        holder.tvErrors.setText(MessageFormat.format("{0} {1}/3",
            getString(R.string.text_view_errors), numErrors));
    else holder.tvErrors.setText(MessageFormat.format("{0} {1}",
        getString(R.string.text_view_errors), numErrors));

    if (numErrors == 3 && getPreferences.isErrorLimiter()) {
        setEndGame();
    }
}
```


Tali metodi vengono chiamati da degli *Observer* implementati per ogni *MutableLiveData*. A questo scopo è stato implementato una classe *SetObserver* nell'activity che permette di inizializzare gli *Observer* utilizzati.

L'utente ha la possibilità di mettere in pausa il gioco ed al click del bottone pausa viene chiamato il metodo *onOptionsItemSelected*, che oscura la griglia mostrando al giocatore un tasto "PLAY" per riprendere il gioco. È possibile riprendere il gioco anche cliccando sulla griglia oscurata.

Se l'utente preme "indietro" viene chiamato il metodo *onBackPressed* che mostra all'utente il dialog di avviso di uscita dalla partita.

```
@Override
public void onBackPressed() {
    if (inGame) {
        if (getPreferences.isDialogShow()) {
            ExitDialog exitDialog = new ExitDialog(this);
            exitDialog.startDialog();
        }
        else {
            getGame();
            restartApp.restart();
        }
    }
    else endGame();
}
```

Viene mostrato lo stesso dialog se viene premuto il tasto "home" sulla *ActionBar*.

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 13 di 46	

Nei metodi *onPause* e *onResume* viene invece fermato e riavviato il timer:

```
// If the activity goes on pause, the timer will be stopped...
@Override
protected void onPause() {
    super.onPause();
    timer.stop();
    elapsedTime = SystemClock.elapsedRealtime() - timer.getBase();
}

// ...and it will be restarted when the activity will be resumed
@Override
protected void onResume() {
    super.onResume();
    timer.setBase(SystemClock.elapsedRealtime() - elapsedTime);
    timer.start();
}
```


Questo permette di mantenere il timer nel caso in cui l'app venga messa in background.

La classe *Holder* in questa activity si prende carico di gestire l'interfaccia utente, quindi la *Grid* che comprende i *Button* numerati da 1 a 9, i *Button btnNotes*, *btnDelete*, *btnHint*, le *TextView* che mostrano gli errori e il livello di difficoltà selezionato e, infine, il *Chronometer* che tiene traccia del tempo impiegato per risolvere il sudoku.

## 4.5 ResultActivity

Ha il compito di gestire le statistiche dell'utente. Il *PagerAdapter* utilizzato contiene 3 *Fragment* (Easy, Medium, Hard) e, ognuno di essi, mostra all'utente il numero di partite totali, il numero di vittorie, la percentuale di vittoria, il miglior tempo e il tempo medio impiegato per il livello di difficoltà relativo.

```
private void setupViewPager(ViewPager viewPager) {
    pagerAdapter = new PagerAdapter(getSupportFragmentManager());
    pagerAdapter.addFragment(new FragmentEasy(), getString(R.string.tab_text_1));
    pagerAdapter.addFragment(new FragmentMedium(), getString(R.string.tab_text_2));
    pagerAdapter.addFragment(new FragmentHard(), getString(R.string.tab_text_3));
    viewPager.setAdapter(pagerAdapter);
}
```


	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 14 di 46	

## 4.6 SettingsActivity

Le impostazioni utente vengono memorizzate nella *SharedPreferences*: il metodo *onSharedPreferencesChanged* si occupa di tenere traccia dei cambiamenti impostati dall'utente per la modalità notturna e per i suoni, in modo da ricreare l'activity nel caso in cui l'opzione venga aggiornata.

```
@Override
public void onSharedPreferencesChanged(SharedPreferences sharedPreferences, String key){
    if (key.equals("night_mode")) {
        if (getPreferences().isNightModeOn()) {
            AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_YES);
            recreate();
        }
        else {
            AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_NO);
            recreate();
        }
    }

    else if (key.equals("sound_effects")) {
        if (getPreferences().isSoundEffectsOn()) {
            setTheme(R.style.AppTheme);
            recreate();
        }
        else {
            setTheme(R.style.DisabledSound);
            recreate();
        }
    }
}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 15 di 46	

## 4.7 TutorialActivity


Il tutorial viene mostrato all'utente come una sequenza di istruzioni divise in 3 *Fragment*.

Il *FragmentOne* mostra all'utente le regole generali del sudoku, il *FragmentTwo* mostra il funzionamento dei controlli ed il *FragmentThree* mostra le funzionalità dell'interfaccia.

```
private Holder() {
    tabLayout = findViewById(R.id.tabLayout);
    viewPager = findViewById(R.id.viewPager);

    setupViewPager(viewPager);
    tabLayout.setupWithViewPager(viewPager);
    tabLayout.addOnTabSelectedListener(this);
}

private void setupViewPager(ViewPager viewPager) {
    pagerAdapter = new PagerAdapter(getSupportFragmentManager());
    pagerAdapter.addFragment(fragmentTutorialOne, null);
    pagerAdapter.addFragment(fragmentTutorialTwo, null);
    pagerAdapter.addFragment(fragmentTutorialThree, null);
    viewPager.setAdapter(pagerAdapter);
}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 16 di 46	

## 5 Fragment

### 5.1 FragmentResult

I *Fragment* sono suddivisi e mostrano le statistiche in base al livello di difficoltà. I *Result* sono salvati su un apposito database e vengono ottenuti tramite delle query SQL distinte in base alla difficoltà. Ogni *Fragment* comprende, inoltre, un *Button* per ripristinare le statistiche di un determinato livello di difficoltà: vengono eliminate le entry dal database.

### 5.2 FragmentTutorial

Sono presenti tre *Fragment* relativi al tutorial divisi in base all'argomento trattato:

- Nel primo vengono spiegate le regole base del sudoku
- Nel secondo vengono mostrati i controlli
- Nel terzo vengono mostrati gli oggetti dell'interfaccia

Per il tutorial è stata utilizzata la libreria *FancyShowCaseView* (<https://github.com/faruktoptas/FancyShowCaseView>). Ogni *Fragment* ha delle proprie *FancyShowCaseView* ed ognuna di essa permette di focalizzarsi su un punto specifico del display, in modo da catturare l'attenzione del giocatore. È possibile costruire una coda di *FancyShowCaseView* e ad ogni tap sullo schermo viene visualizzata la prossima view.

### 5.3 FragmentSettings

Viene utilizzato un *Fragment* di supporto per mostrare la schermata di impostazione. Il *Fragment* è molto semplice:


```
// Settings fragment to change preferences
public class FragmentSettings extends PreferenceFragmentCompat {

    @Override
    public void onCreatePreferences(Bundle savedInstanceState, String rootKey) {
        addPreferencesFromResource(R.xml.preference_screen);
    }

}
```

È necessario che il *Fragment* estenda la classe *PreferenceFragmentCompat* e che utilizzi un file XML che comprenda le varie opzioni gestibili dall'utente dell'applicazione.



	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 17 di 46	

## 6 Adapter

Nell'app viene utilizzato un *PagerAdapter* per scorrere tra i *Fragment* della *ResultActivity* e *TutorialActivity*. Per inizializzare l'*Adapter* sono necessari una *List* di *Fragment* e i titoli di questi ultimi:

```
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();
    }
}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 18 di 46	

## 7 Classi custom

### 7.1 GetPreferences

Questa classe viene fornita come supporto alle activity per l'accesso alle *SharedPreferences*. Infatti, l'utente modificando le impostazioni va a modificare le variabili tra le *SharedPreferences*.

Vengono messi a disposizione metodi di check:


```
public void checkEnabledSound() {
    if (preferences.getBoolean("sound_effects", true)) {
        context.setTheme(R.style.AppTheme);
    }
    else {
        context.setTheme(R.style.DisabledSound);
    }
}

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

public void checkNightMode() {
    if (preferences.getBoolean("night_mode", false)) {
        AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_YES);
    } else {
        AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_NO);
    }
}
```

E metodi di get per accedere alle variabili delle impostazioni, impostate in base alle preferenze dell'utente:

```
public boolean isTimerShow() {return preferences.getBoolean("timer", true);}
public boolean isSolverShow() {return preferences.getBoolean("solver", true);}
public boolean isErrorLimiter(){return preferences.getBoolean("error_limiter", false);}
public boolean isHintShow() {return preferences.getBoolean("step", true);}
public boolean isDialogShow() {return preferences.getBoolean("dialog", true);}
public boolean isNightModeOn() {return preferences.getBoolean("night_mode", false);}
public boolean isSoundEffectsOn(){return preferences.getBoolean("sound_effects", true)}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 19 di 46	

## 7.2 RestartApp

Viene fornito il metodo *restart* che crea una nuova *Intent*, in cui vengono impostati dei flag per pulire lo stack delle activity, e riavvia l'app senza mostrare lo splash screen:

```
public void restart() {
    Intent i = new Intent(activity, MainActivity.class);
    i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);
    i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    activity.finish();
    activity.startActivity(i);
}
```

## 7.3 DataConverter

Convertitore *List<Cell>/String* che utilizza la libreria Gson. Utilizzato dal database *Game*, in quanto la libreria *Room* non supporta il salvataggio di classi custom (come *Cell*).

```
@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);
}


@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);
}
```

## 7.4 TimeParser

Parser da *Long* a *String* in formato "min:sec":

```
public String parse(long time) {
    @SuppressWarnings("DefaultLocale")
    String ret = String.format("%02d:%02d",
        TimeUnit.MILLISECONDS.toMinutes(time) -
            TimeUnit.HOURS.toMinutes(TimeUnit.MILLISECONDS.toHours(time)),
        TimeUnit.MILLISECONDS.toSeconds(time) -
            TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(time)));
    return ret;
}
```

Usata per mostrare il tempo (espresso come *Long*) nelle *TextView* (che necessita di *String*).


	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>20</b> di <b>46</b>	

## 8 Dialog

Nell'applicazione sono disponibili molti *Dialog* che forniscono varie informazioni all'utente. Ogni *Dialog* ha una propria classe di gestione.

I *Dialog* più interessanti da analizzare sono:

- *BottomDialog*: esso, oltre i metodi di show e di dismiss, comprende anche un proprio *Holder*. Quest'ultimo è necessario per gestire le view del layout del *Dialog*, infatti, esse devono essere ricreate ad ogni show del *Dialog*.
- *NoInternetDialog*: informa l'utente della mancata connessione alla rete. Per dispositivi con una versione di Android minore di Q (Android 10) verrà data, all'utente, la possibilità di abilitare il Wi-Fi.
- *LoadingDialog*: viene mostrato quando l'app è in attesa del completamento del download della griglia e viene rimosso solo quando la schermata di gioco viene caricata.

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 21 di 46	

## 9 ViewModel


La classe *PlaySudokuViewModel* fornisce il *ViewModel* per la view *SudokuBoardView*, una per una nuova partita, ed una per ripristinare l'ultima partita in corso. La view *SudokuBoardView* corrispondente si occupa invece di disegnare la griglia di gioco.

```
public class PlaySudokuViewModel extends ViewModel {
    private SudokuGame sudokuGame;

    // For new game
    public PlaySudokuViewModel(ArrayList<Integer> board) {
        sudokuGame = new SudokuGame(board);
    }

    // For in progress game
    public PlaySudokuViewModel(List<Cell> cells, List<Cell> solvedCells, int numErr) {
        sudokuGame = new SudokuGame(cells, solvedCells, numErr);
    }

    public SudokuGame getSudokuGame() {
        return sudokuGame;
    }
}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 22 di 46	

## 10 Database

La libreria utilizzata per salvare i dati è *Room* che semplifica l'accesso al database *SQLite*.

L'entità *Result* memorizza le statistiche dell'utente:

```
@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;
}
```


...

Mentre l'entità *Game* permette di salvare i dati relativi all'ultima partita in corso:

```
@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;
}
```

...

Entrambe le entità hanno un proprio *DAO* che permette di eseguire le query relative alle due tabelle.

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 23 di 46	

## 11Volley

Fornisce 3 metodi di download delle griglie del sudoku in base al livello di difficoltà scelto dall'utente:

```
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);
}
```


Un metodo *apiCall* per inserire una richiesta HTTP nella coda delle richieste:

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

Ed un metodo *onResponse* che converte il JSON ottenuto in un *ArrayList<Integer>* e crea una nuova Intent per avviare la *PlaySudokuActivity* in cui inserisce la griglia ottenuta, la difficoltà scelta e imposta la variabile "new\_game" a true:

```
@Override
public void onResponse(String response) {
    ArrayList<Integer> board;
    try {
        JSONObject jsonObject = new JSONObject(response);
        JSONArray jsonArray = jsonObject.getJSONArray("board");
        board = convert(jsonArray);

        Intent intent = new Intent(activity, PlaySudokuActivity.class);
        intent.putExtra("board", board);
        intent.putExtra("difficulty", difficulty);
        intent.putExtra("new_game", false);
        activity.startActivity(intent);
    }
    catch (JSONException e) {
        e.printStackTrace();
    }
}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>24</b> di <b>46</b>	

## 12 Game

Nel package *com.mp.sudoku.game* ci sono 3 classi che implementano il vero e proprio gioco:

- La classe *Board* che contiene una *List<Cell>* delle *Cell* contenute nella griglia di gioco
- La classe *Cell* che identifica una singola cella di gioco
- La classe *SudokuGame* che contiene la logica completa del gioco del sudoku

### 12.1 Board

Contiene un intero che indica il numero di blocchi della griglia (di standard è impostata a 9), e una *List* di *Cell* contenute nella griglia di gioco. Il costruttore principale prende come input una dimensione e una *List<Cell>*:

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


Inoltre, sono disponibili i vari metodi di set e di get.

### 12.2 Cell

Contiene vari oggetti:

- *Solve*: booleano utilizzato per indicare se una *Cell* fa parte della griglia risolta
- *Row*: intero che indica la riga della *Cell*
- *Col*: intero che indica la colonna della *Cell*
- *Value*: intero da 1 a 9 che indica il valore della *Cell*
- *Valid*: booleano che indica se una *Cell* è valida
- *IsStartingCell*: booleano che indica se una *Cell* fa parte della griglia di partenza ottenuta con la *Volley*
- *Notes*: lista di interi relativi alle note inserite per la *Cell*



	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 25 di 46	

Il costruttore principale prende come input i valori appena descritti, tranne *Valid*. Quest' ultimo viene impostato durante la partita dalla classe *SudokuGame*:

```
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);
    }
}
```

Inoltre, sono disponibili i vari metodi di set e di get.


## 12.3 SudokuGame

Contiene tutta la logica di gioco. I metodi principali sono:

- *solveBoard*, restituisce una nuova *Board* risolta:

```
// Solve method
private boolean solveBoard() {
    for (Cell cell : solvedBoard.getCells()) {
        if (cell.getValue() == 0) {
            cell.setValid(true);
            for (int number = 1; number <= 9; number++) {
                if (isOk(cell.getRow(), cell.getCol(), number)) {
                    cell.setValue(number);

                    if (solveBoard()) {
                        return true;
                    }
                } else {
                    cell.setValue(0);
                }
            }
        }
        return false;
    }
    return true;
}
```


	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 26 di 46	

- *reset*, ripristina la griglia di gioco allo stato iniziale:

```
// Reset Sudoku method
public void reset() {
    for (Cell cell : board.getCells()) {
        if (!cell.isStartingCell()) {
            cell.setValue(0);
            cell.setValid(false);
            cell.setNotes(new ArrayList<Integer>());
        }
    }
    cellsLiveData.postValue(board.getCells());
}
```

- *hint*, inserisce una *Cell* della *SolvedBoard* nella griglia di gioco:

```
// Hint method that set a correct CELL in the game BOARD
public void hint() {
    useHint = true;
    Random rand = new Random();
    Cell randomCell =
solvedBoard.getCells().get(rand.nextInt(solvedBoard.getCells().size()));
    Cell cell = board.getCell(randomCell.getRow(), randomCell.getCol());
    if (cell.getValue() == randomCell.getValue()) {
        hint();
    }
    else {
        cell.setValue(randomCell.getValue());
        cell.setValid(true);
        cell.setStartingCell(true);
        cellsLiveData.postValue(board.getCells());
        endGame();
    }
}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 27 di 46	

- *validateCell*, effettua un controllo sulla validità della *Cell*:

```
// Validate a CELL with a check on all the others
private void validateCell(Cell cell) {
    int r = cell.getRow();
    int c = cell.getCol();
    int sqrtSize = 3;


    for (Cell tempCell : board.getCells()) {
        if (cell.getValue() != 0) {

            if (cell.getValue() == tempCell.getValue() &&
                (r == tempCell.getRow() || c == tempCell.getCol()
                 || (r / sqrtSize == tempCell.getRow() / sqrtSize
                    && c / sqrtSize == tempCell.getCol() / sqrtSize))
                && cell != tempCell) {

                cell.setValid(false);
                tempCell.setValid(false);
                return;
            } else {
                cell.setValid(true);
            }
        }
    }
}
```

Sono disponibili due costruttori: uno per una nuova partita ed uno per una partita già in corso. La differenza tra i due è data dagli input:

- Il primo ha come input un *ArrayList* di interi
- Il secondo ha come input una *List* di *Cell*

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 28 di 46	

## 13 View

Per la corretta fruizione dell'app è stato necessario creare una custom view che rappresenta graficamente la griglia del sudoku. La *SudokuBoardView* comprende una serie di *Paint* utilizzati per disegnare le linee che compongono la griglia e i numeri che appaiono nelle caselle. La view misura le dimensioni dello schermo del dispositivo attraverso il metodo *onMeasure*:

```
// It returns the BOARD measures according to the device display
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec);
    int sizePixels = Math.min(widthMeasureSpec, heightMeasureSpec);
    setMeasuredDimension(sizePixels, sizePixels);
}
```

E, attraverso tali misure, viene disegnata una griglia adatta al display. Attraverso la classe *GetPreferences*, è possibile accedere alle opzioni dell'utente relative alle impostazioni grafiche dell'app. Le *Paint* della view vengono disegnate in base a tali impostazioni.

All'interno della view sono disponibili una serie di metodi per oscurare la view:

```
// Obscure the BOARD
public void obscureView() {
    visibility = false;
    invalidate();
}
```

Per schiarire la view:


```
// Reveal the BOARD
public void revealView() {
    visibility = true;
    invalidate();
}
```

Per aggiornare i valori della cella selezionata:

```
// Update Selected CELL values
public void updateSelectedCellUI(Cell cell) {
    selectedCell = cell;
    selectedRow = cell.getRow();
    selectedCol = cell.getCol();
    invalidate();
}
```

Per aggiornare i valori delle celle della griglia:

```
// Update all CELLS
public void updateCells(List<Cell> cells) {
    this.cells = cells;
    invalidate();
}
```


	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 29 di 46	

Inoltre, nella view è stata inserita l' interfaccia *OnTouchListener*. Essa è un listener implementato nella *PlaySudokuActivity* che ha come metodo *onCellTouched*. Lo scopo di tale listener è quello di creare un'interfaccia tra l'activity e la view, per aggiornare i valori della cella selezionata (toccata):

```
// Interface to handle TAP on a CELL
public interface OnTouchListener {
    void onCellTouched(int row, int col);
}
```

Infine, è presente il metodo *onTouchEvent*, per rilevare il tap su una cella e informare il listener di tale evento:


```
// Detects tap on a CELL
@Override
public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        handleTouchEvent(event.getX(), event.getY());
        return true;
    }
    else {
        return false;
    }
}
```

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>30</b> di <b>46</b>	

## 14 Drawable

Tutti i drawable utilizzati sono dei *Vector Assets*. Tale scelta è stata effettuata per garantire una migliore qualità visiva delle immagini in tutti i display. Le uniche immagini in formato PNG sono quelle relative al tutorial. Tutti i drawable utilizzati:

- Fanno parte di default di Android Studio
- Sono stati scaricati da IconMonstr (<https://iconmonstr.com/>)


	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>31</b> di <b>46</b>	

## 15 Colors

Per implementare la Night mode, è stato necessario inserire due file XML (colors):

- Uno per la modalità Light
- Uno per la modalità Night

I due file definiscono quali colori utilizzare per le due modalità grafiche.

	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>32</b> di <b>46</b>	


## 16 Strings

Nell'applicazione è stata implementato il supporto per la lingua inglese e la lingua italiana. Sono disponibili due file XML (strings):

- Uno per la lingua inglese
- Uno per la lingua italiana

I due file definiscono quali stringhe utilizzare in base alla lingua del dispositivo.




	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 33 di 46	

## 17 Layout

### 17.1 MainActivityLayout

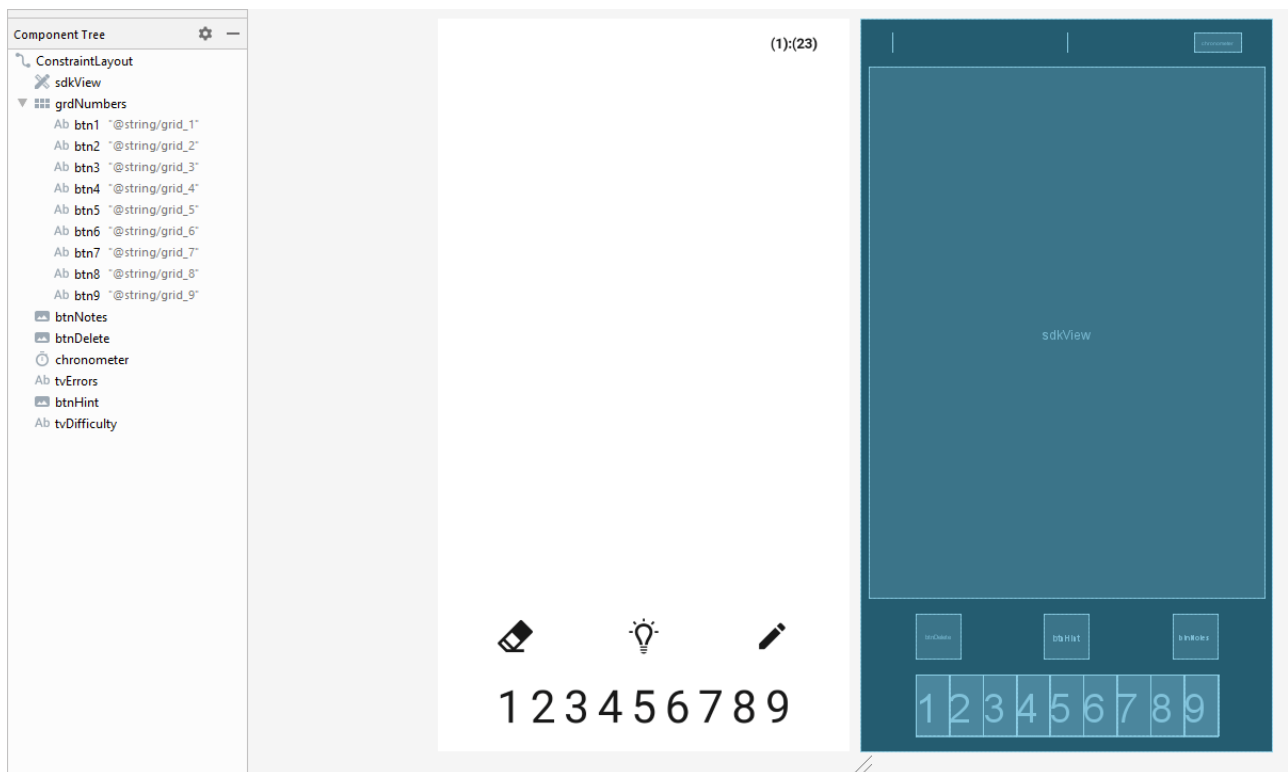
*MainActivityLayout* ha come root un *ConstraintLayout* e dispone di un *Button* “Play”, un *Button* “Continue” nascosto che verrà visualizzato solo quando nel database è presente una partita in corso. Inoltre, questo bottone mostrerà al suo interno i dettagli della partita salvata. Il Layout comprende anche una barra *bottom\_navigation* per accedere alle statistiche di gioco o tornare alla Home, delle *ImageView* di decoro ed il logo dell’applicazione.



	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 34 di 46	

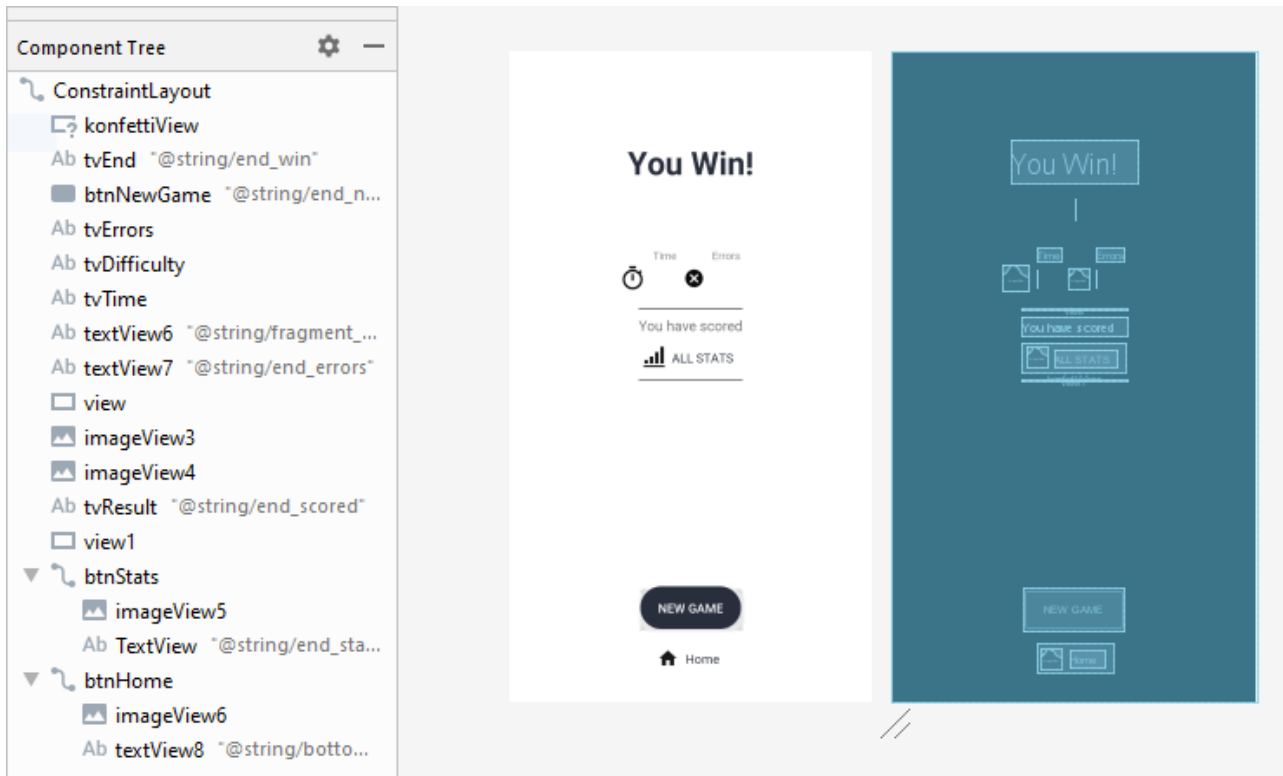
## 17.2 PlaySudokuActivityLayout


Il *PlaySudokuActivityLayout* è il layout di gioco e comprende una *Grid* di numeri, i *Button* Notes, Delete ed Hint, un *Chronometer* per visualizzare il timer di gioco, una *TextView* per indicare il livello di difficoltà ed una che visualizza a schermo il numero di Errori ottenuti dal giocatore nella partita corrente. Infine, comprende una *SudokuBoardView*, che permette di visualizzare graficamente la griglia di gioco.



## 17.3 EndGameActivityLayout

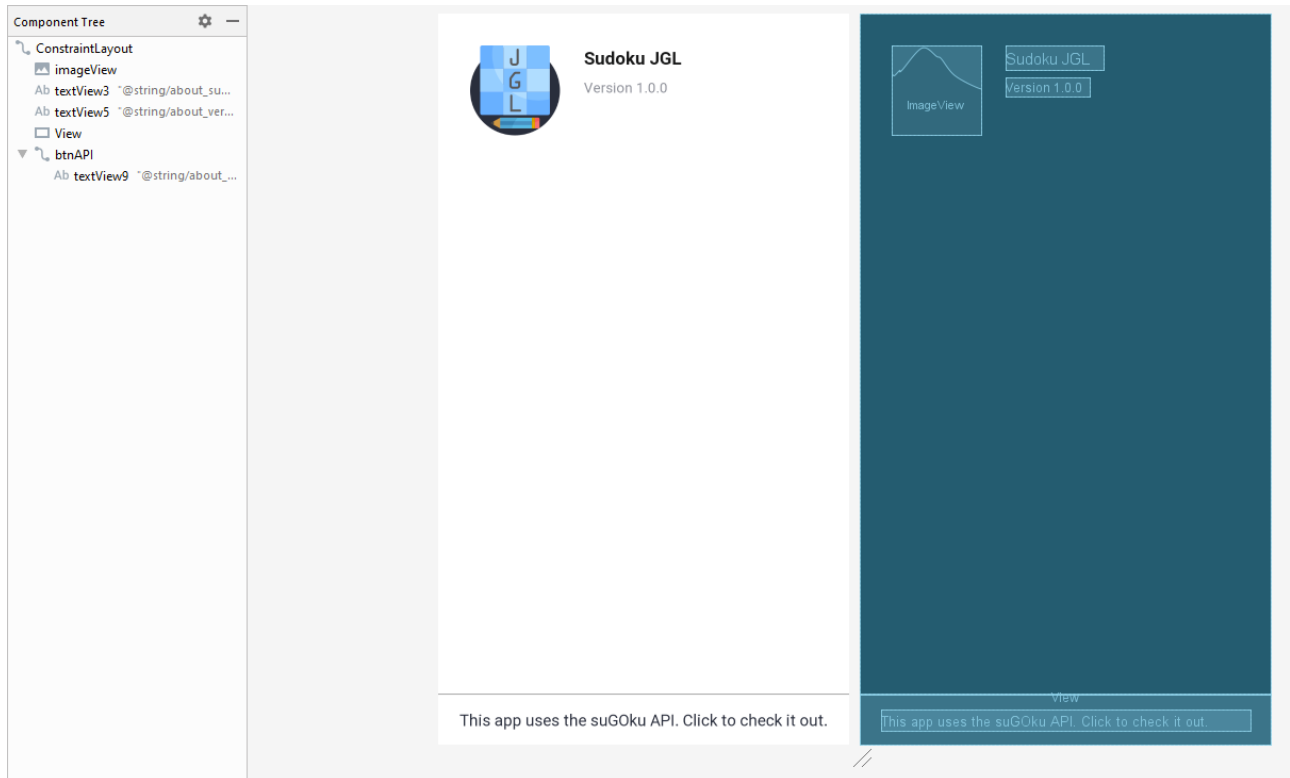
Al termine della partita viene mostrata all'utente una schermata di fine gioco con le varie statistiche nelle *TextView*, lo status finale della partita (win/lose), un *Button* per accedere alle statistiche, un *Button* per iniziare una nuova partita ed un *Button* per tornare alla Home.




	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 36 di 46	

## 17.4 AboutActivityLayout

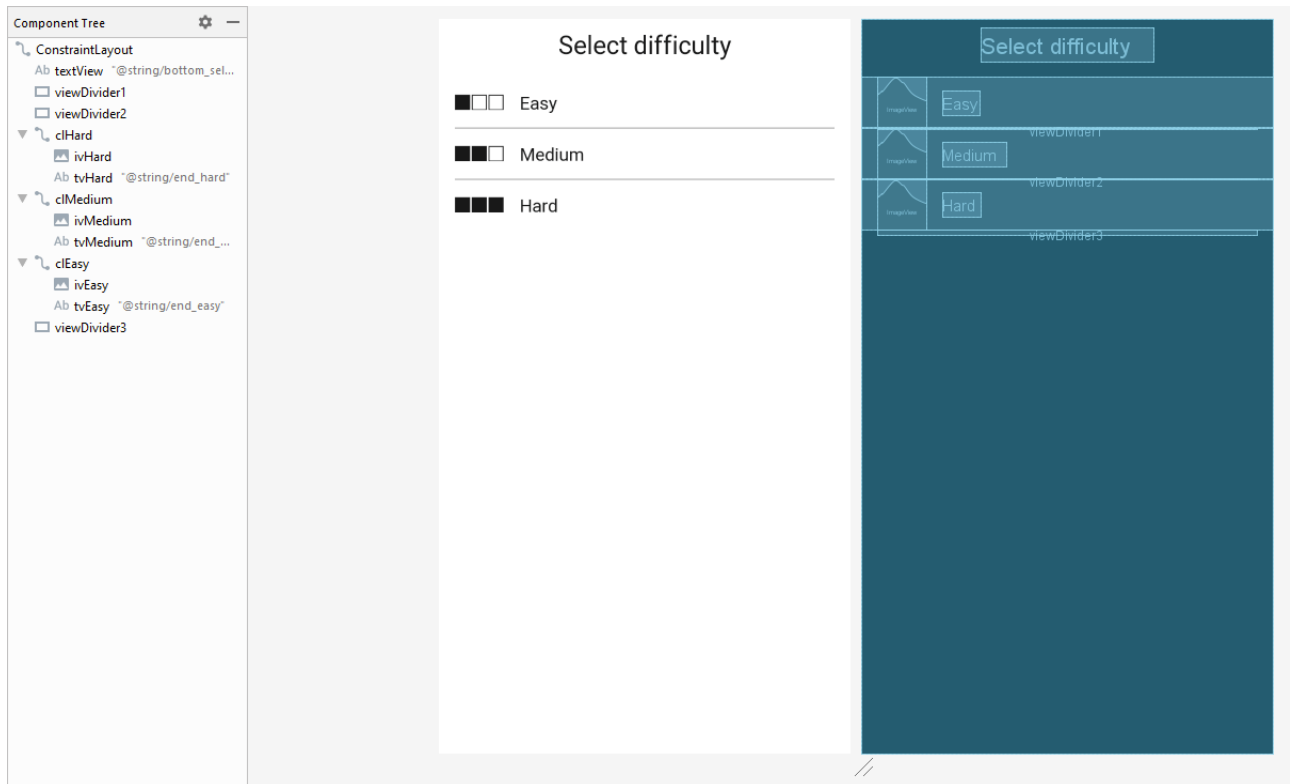
È un *Layout* molto semplice: comprende una *ImageView* con il logo dell'app, due *TextView*, una per il nome dell'applicazione ed una per indicarne la versione, e un *ConstraintLayout* cliccabile, per visualizzare il sito delle API utilizzate nell'app.




	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 37 di 46	

## 17.5 BottomSheetLayout

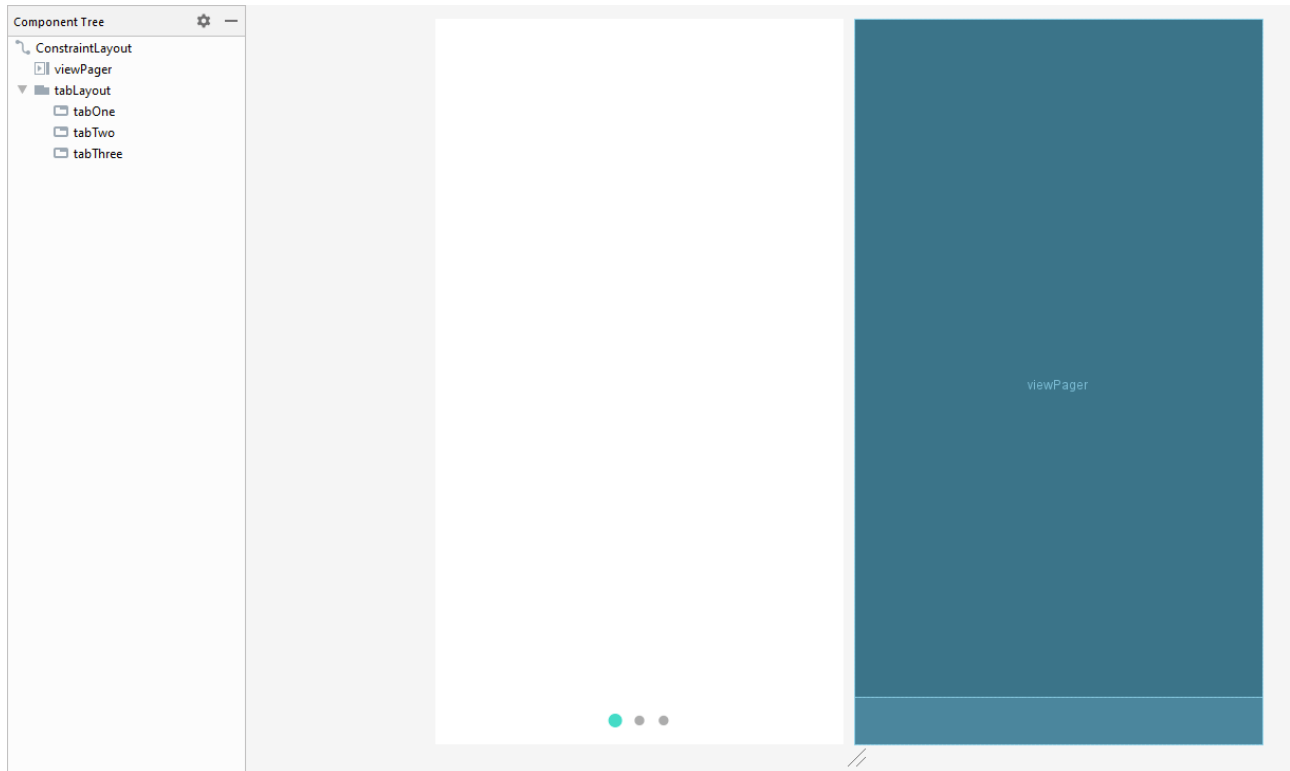
È il *Layout* utilizzato dal *BottomDialog* che permette la scelta del livello di difficoltà. Infatti, comprende tre *ConstraintLayout*, cliccabili, ognuno per una delle difficoltà.




	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>38</b> di <b>46</b>	

## 17.6 TutorialActivityLayout

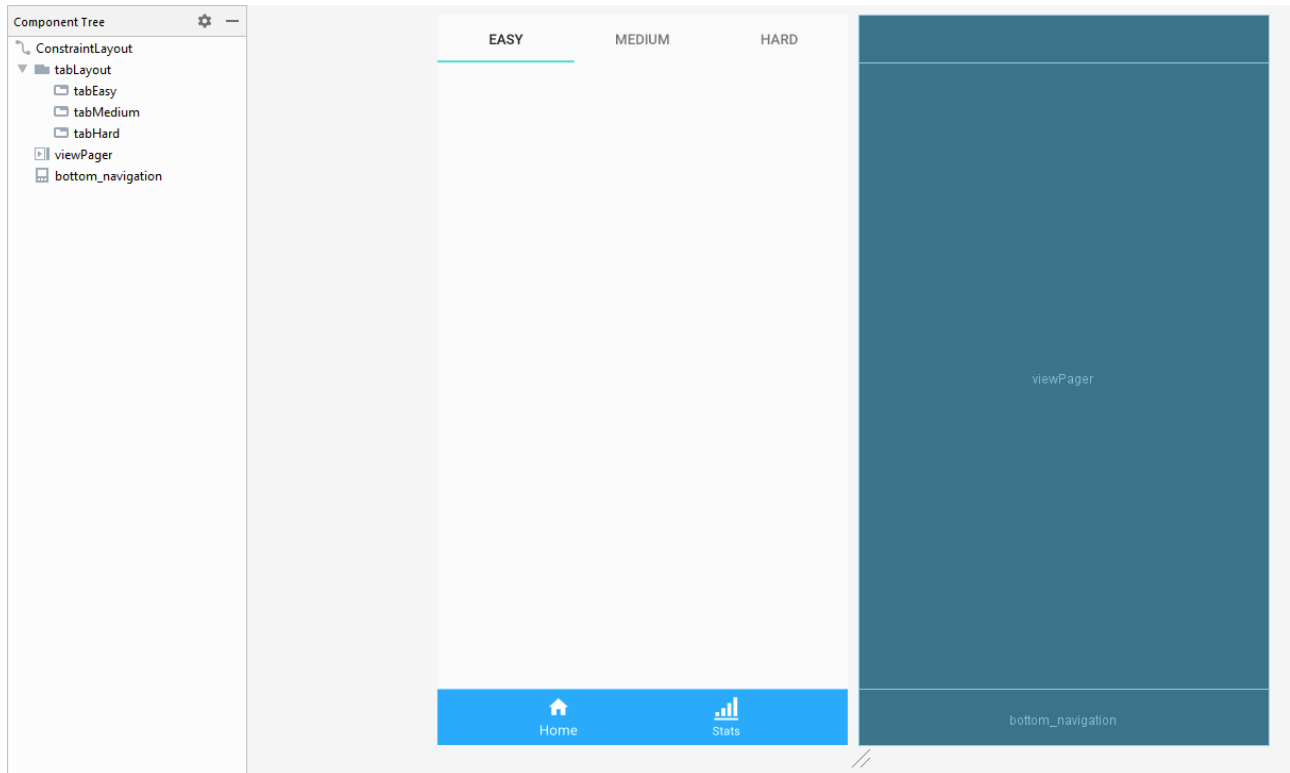
Il *Layout* comprende un *TabLayout*, con tre schede (una per ogni schermata del tutorial), e un *ViewPager*, per scorrere tra i *Fragment*.



	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>39</b> di <b>46</b>	

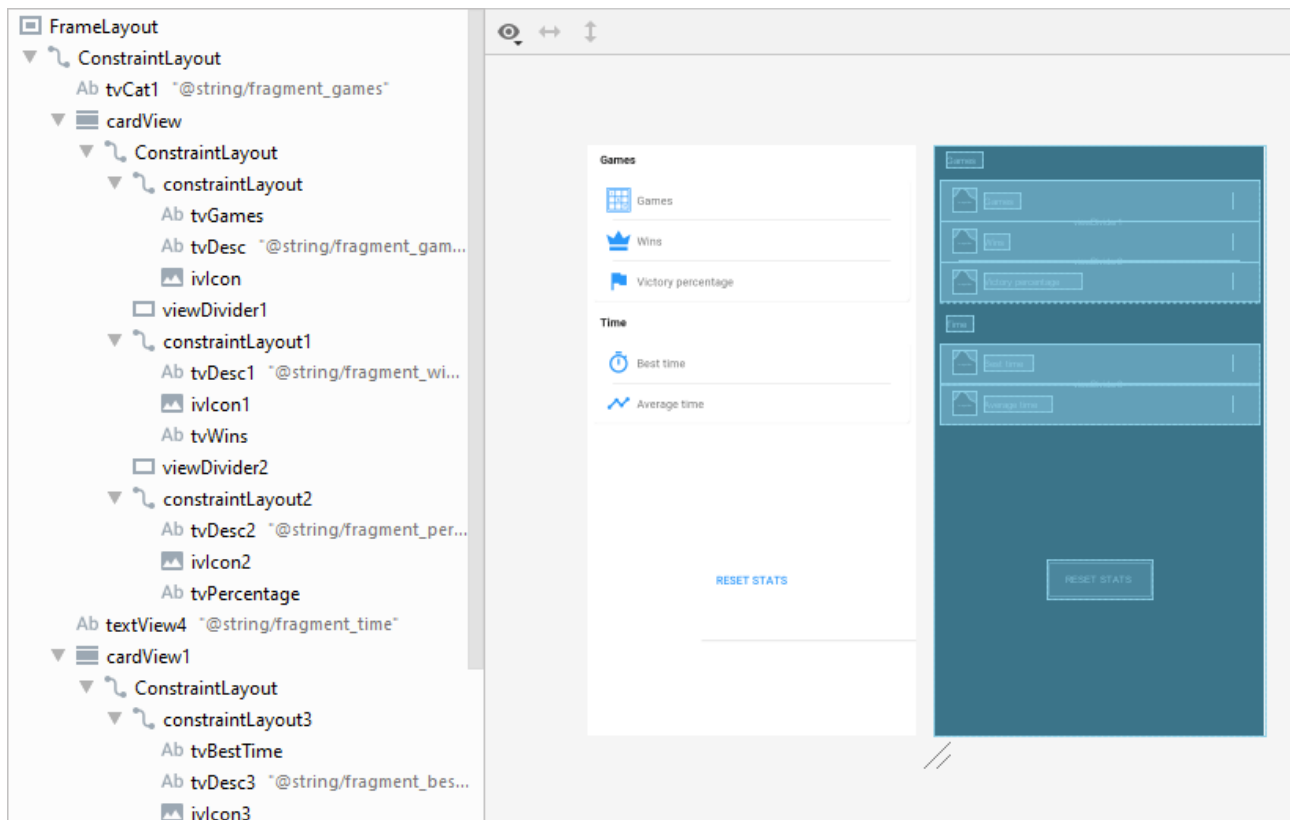
## 17.7 ResultActivityLayout

Comprende un *TabLayout* in alto, con tre schede (una per ogni livello di difficoltà), un *ViewPager*, per scorrere tra i *Fragment*, e una *BottomNavigationBar*, per poter tornare alla Home.



## 17.8 ResultFragmentLayout

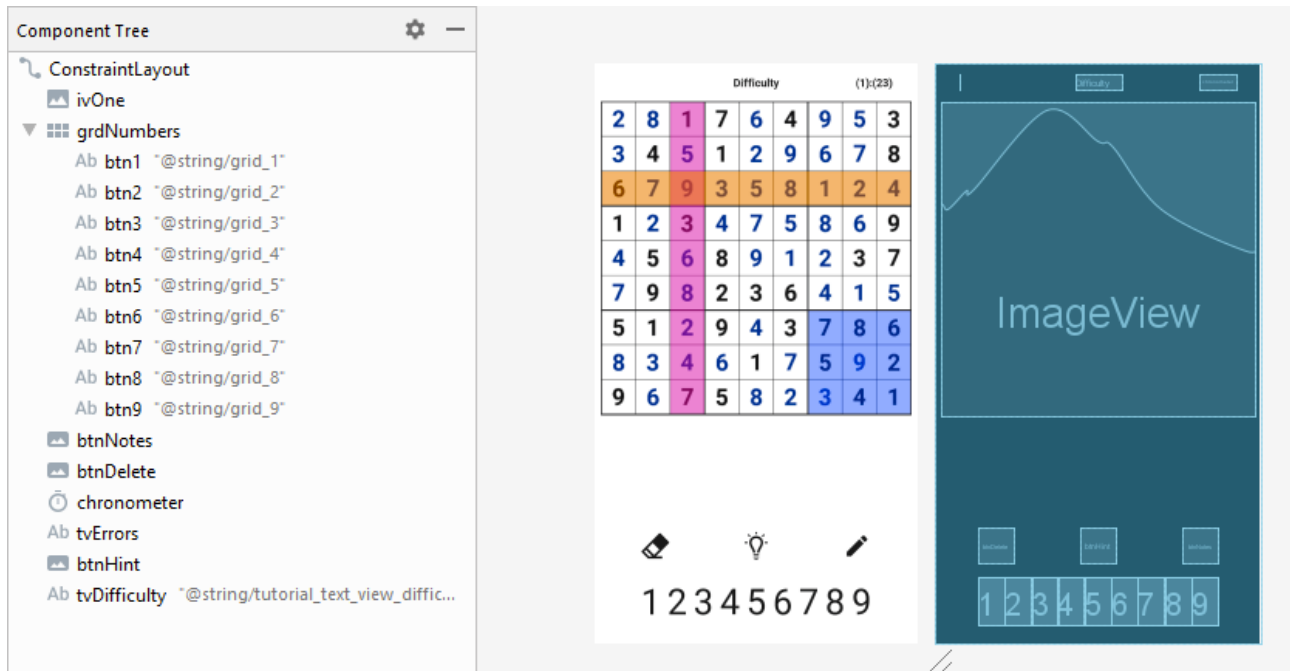
Le statistiche di gioco sono divise per livello di difficoltà tramite dei *Fragment* che utilizzano, come layout, il *FragmentResultLayout* in cui le singole statistiche sono inserite in due *CardView*. È possibile per l'utente resettare le statistiche di gioco premendo il *Button* "RESET STATS".





## 17.9 TutorialFragmentLayout

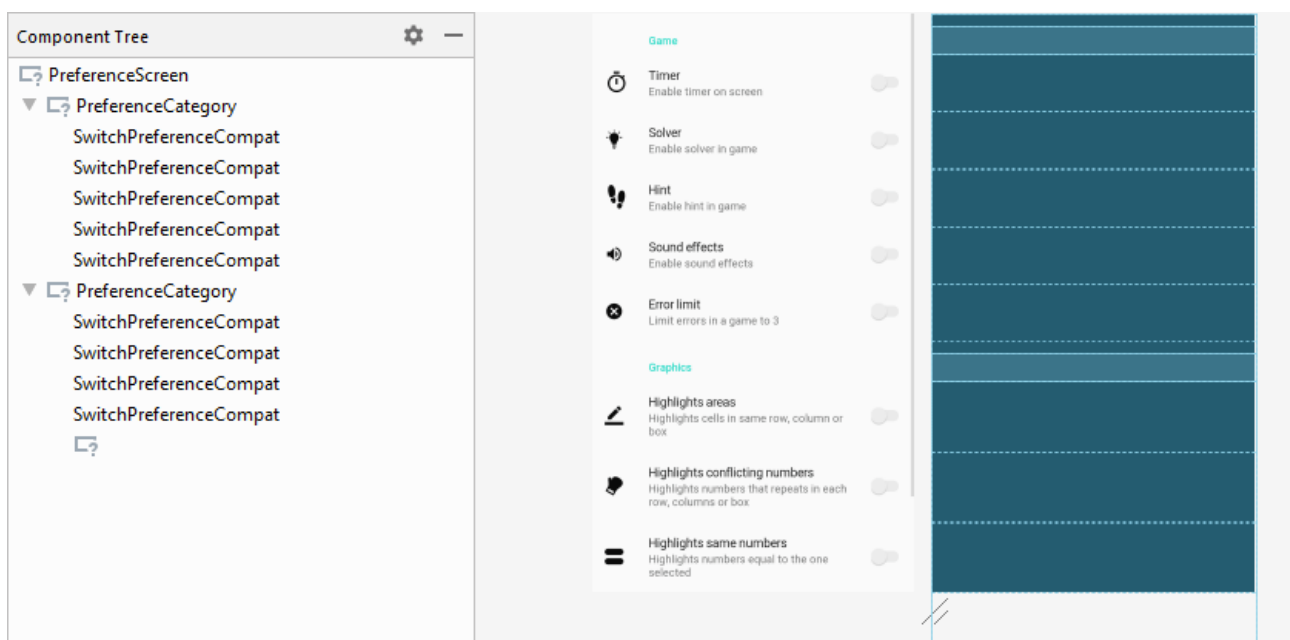
Il layout *FragmentTutorialLayout* è utilizzato dai *Fragment* del tutorial che mostra all'utente una panoramica di utilizzo dell'applicazione ed è identico al *PlaySudokuActivityLayout*. Differiscono solo per il fatto che in questo layout la *SudokuBoardView* è sostituita da una *ImageView*.



## 17.10 PreferenceScreenLayout

Viene utilizzato un *PreferenceScreen* il quale mette a disposizione una *ListView* editabile, con la possibilità di inserire vari oggetti per gestire le impostazioni. In questo layout sono stati inseriti degli *Switch* e un *CheckBox*. Per ognuno di essi è stata inserita un' icona, un titolo e una descrizione. Inoltre, le varie opzioni sono state divise in due categorie:


- Gioco
- Grafica



## 17.11 FancyShowCaseViewLayout

Sono disponibili diversi *Layout* per le *FancyShowCaseView* in base alla posizione del testo:

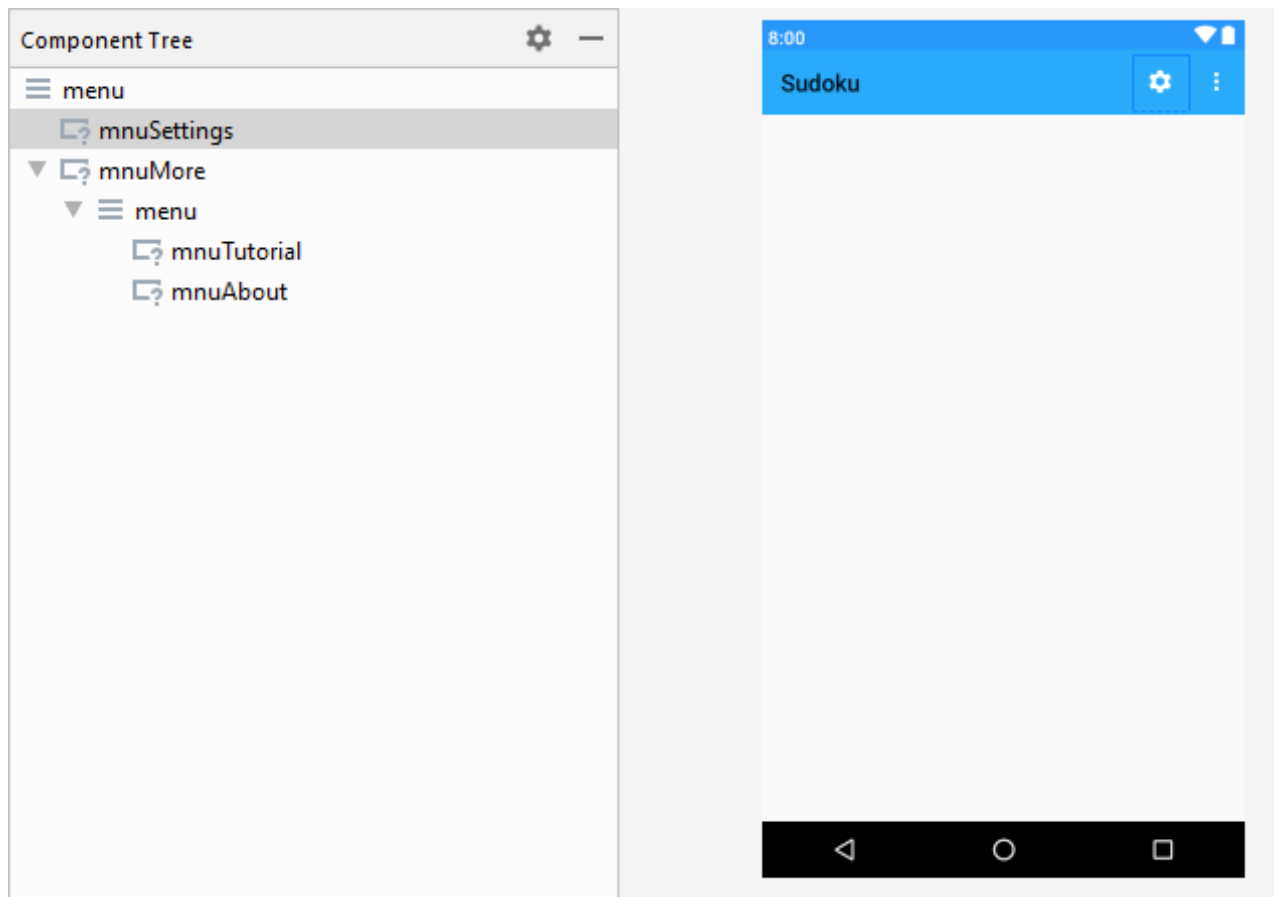
- Testo centrato: per le informazioni generali
- Testo in basso: per le informazioni relative agli oggetti in basso nello schermo
- Testo in alto: per le informazioni relative agli oggetti in alto nello schermo


	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>43</b> di <b>46</b>	

## 18 Menu

### 18.1 MainActivityMenu

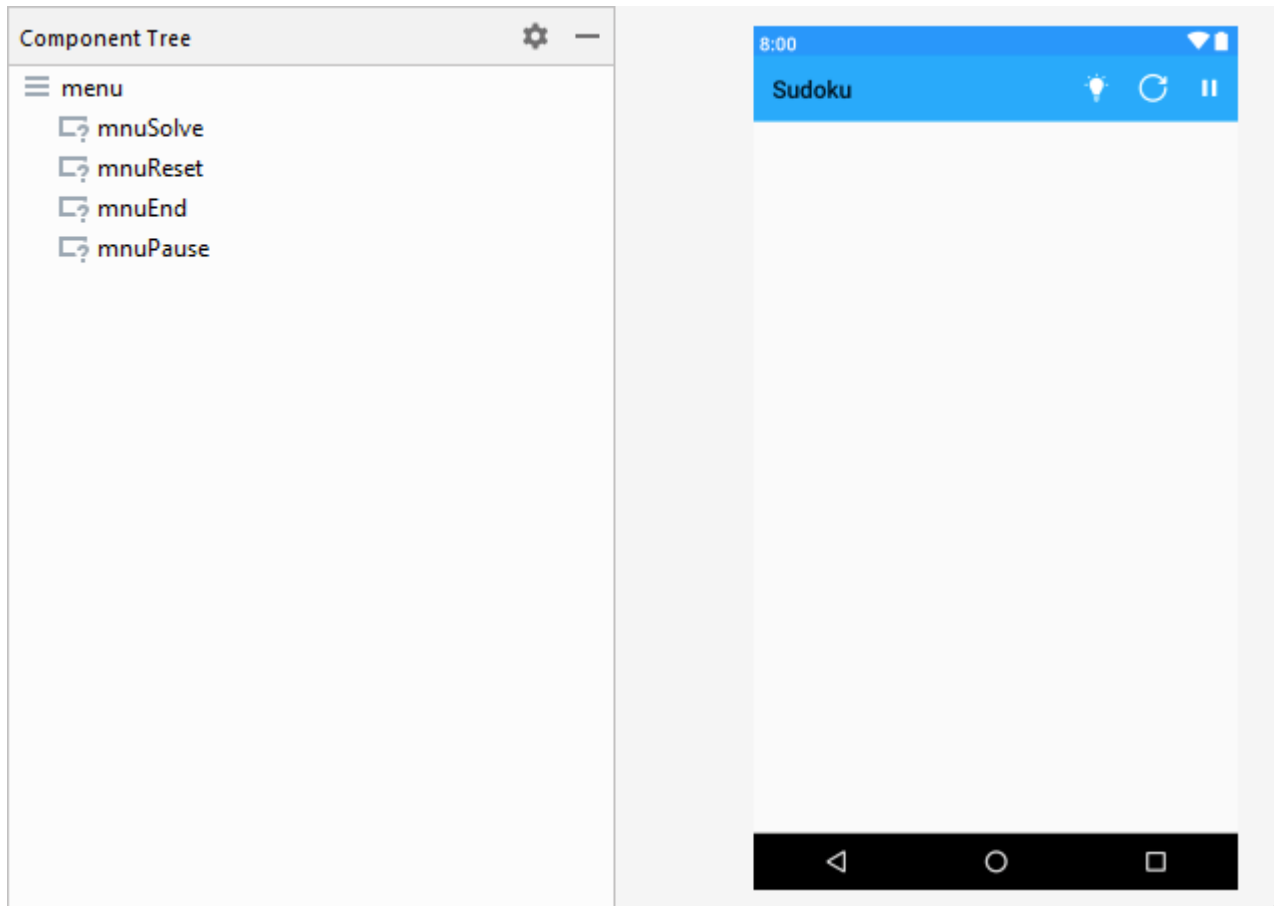
Il *Menu* della *MainActivity* mette a disposizione dell'utente un item di impostazioni ed un item more, che contiene altri due item: uno per l'accesso al tutorial ed uno per visualizzare le info sull'applicazione.




	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>44</b> di <b>46</b>	

## 18.2 PlaySudokuActivityMenu

*Menu* disponibile durante il gioco, che permette all'utente di mettere in pausa la partita, risolvere completamente il sudoku , uscire dalla partita o ripristinare il sudoku corrente.

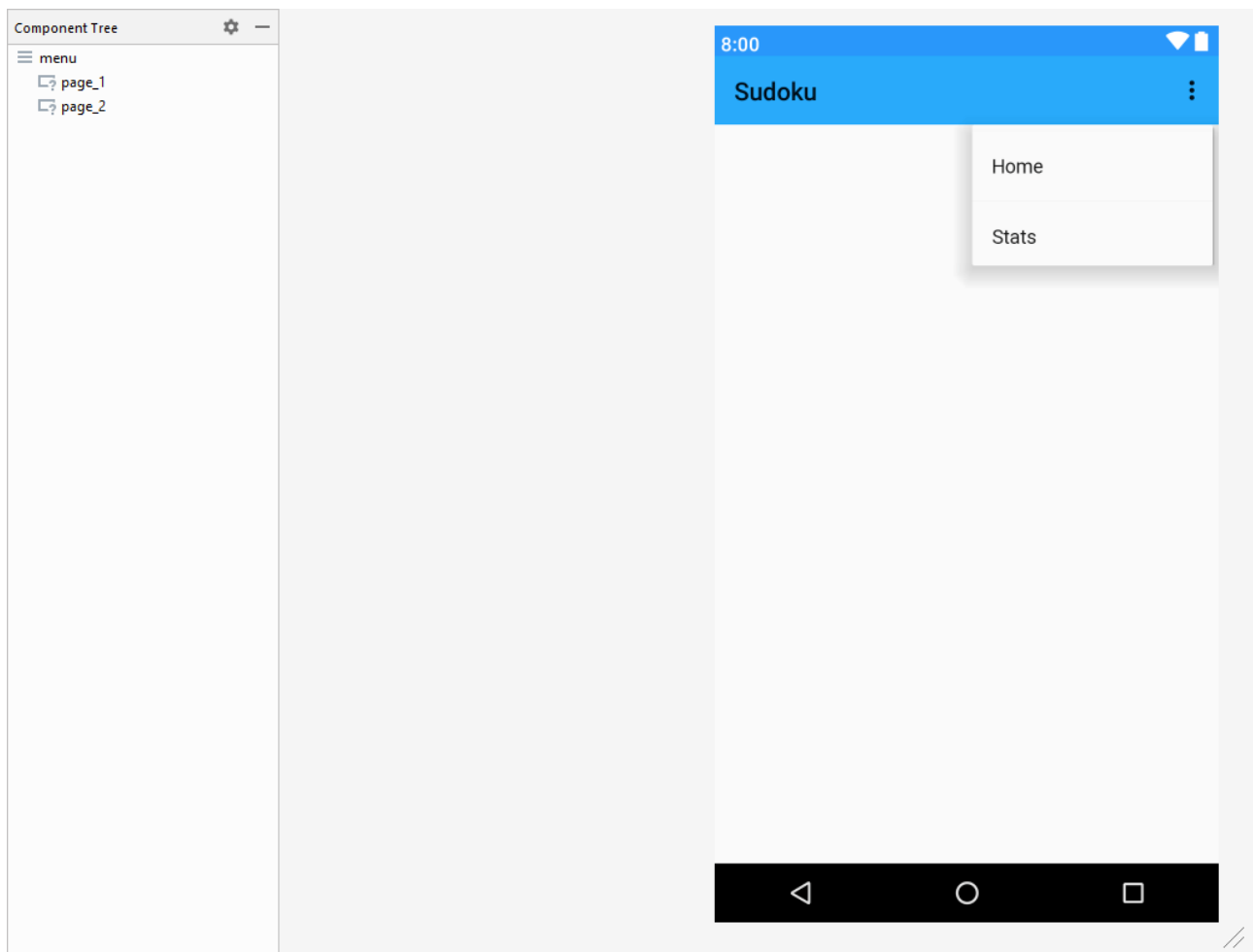



	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. <b>45</b> di <b>46</b>	

## 18.3 BottomNavigationBar

È un *Menu* molto semplice ed è utilizzato per la barra di navigazione inferiore per la *MainActivity* e la *ResultActivity*. Comprende due item:

- Home: per tornare alla *MainActivity*
- Stats: per avviare la *ResultActivity*



	JGL Sudoku	Rev.	1.0
		Data	15.06.2020
		Pag. 46 di 46	

## 19 Style & Locale

L'app viene fornita in lingua inglese di default e in lingua italiana.

Gli stili utilizzati sono quelli della modalità *Light*:

- *AppTheme*: tema di base utilizzato dall'app
- *DisabledSound*: per disabilitare i suoni dell'applicazione
- *SplashTheme*: utilizzato dalla schermata splash mostrata all'avvio dell'app
- *FancyShowTitle*: utilizzata per le *TextView* dei layout relativi alle *FancyShowCaseView* del tutorial

```
<resources>
  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
  </style>

  <!-- Disabled sound effects theme-->
  <style name="DisabledSound" parent="AppTheme">
    <item name="android:soundEffectsEnabled">false</item>
  </style>

  <!-- Splash theme-->
  <style name="SplashTheme" parent="Theme.AppCompat.NoActionBar">
    <item name="android:windowBackground">@drawable/background_splash</item>
    <item name="android:windowNoTitle">true</item>
    <item name="android:windowActionBar">false</item>
    <item name="android:windowFullscreen">true</item>
    <item name="android:windowContentOverlay">@null</item>
  </style>

  <!-- Fancy text tutorial style-->
  <style name="FancyShowTitle" parent="Widget.AppCompat.TextView">
    <item name="android:textSize">24sp</item>
    <item name="android:gravity">center</item>
    <item name="android:textColor">@color/fancyText</item>
    <item name="fontFamily">sans-serif-medium</item>
  </style>
</resources>
```

Inoltre, è disponibile uno stile di base per la modalità *Night*:

```
<resources>
  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.DayNight.DarkActionBar">
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
  </style>
</resources>
```