



MIT APP INVENTOR

1

di FRANCESCO FICILI

Iniziamo a scoprire MIT App Inventor, un tool di sviluppo per applicazioni Android creato dal MIT e basato sul linguaggio di programmazione grafico Scratch. In questa prima puntata cercheremo di prendere confidenza con l'ambiente di sviluppo e creeremo la nostra prima, semplice app. Prima puntata.

Le applicazioni per smartphone sono diventate una realtà dei nostri giorni, al punto che ormai ne scarichiamo ed utilizziamo una quantità considerevole per soddisfare le esigenze più disparate. In particolare, con l'avvento e la diffusione dei sistemi basati su Android, il numero di app è ulteriormente aumentato, trattandosi di applicazioni ancora più semplici da realizzare rispetto alla controparte iOS, vuoi per la maggiore semplicità della pubblicazione di una app sul Play Store rispetto all'Apple Store, vuoi per le caratteristiche dei linguaggi utilizzati e delle piattaforme di sviluppo. Cionondimeno, la realizzazione di una app Android richiede ancora delle competenze specifi-



Fig. 1 - Logo di Android.



Fig. 2 - Logo di MIT App Inventor.

che di programmazione, come la conoscenza del linguaggio Java ed una certa confidenza con il framework Android.

Per ovviare a questa problematica è stata sviluppata una serie di tools che permettono di sviluppare semplicemente app Android senza la necessità di essere dei programmatori Java provetti e conoscere dettagliatamente ambienti di sviluppo complessi. Uno dei tool di questo tipo più utilizzati (conta ad oggi circa 2 milioni di utenti sparsi per il mondo) è MIT App Inventor.

Introduzione a MIT App Inventor

Sviluppato originariamente da Google, nel 2011 il progetto è passato sotto la gestione del MIT (Mas-

sachusetts Institute of Technology), prendendo la denominazione di MIT App Inventor EDU, o, più comunemente, MIT App Inventor; il relativo logo è riportato in Fig. 2.

L'uso di MIT App Inventor rende la realizzazione di una app Android un processo molto più intuitivo rispetto ai metodi tradizionali basati su linguaggi testuali; infatti per realizzare una app con App Inventor si fa uso di una serie di blocchi simili a pezzi di un puzzle per la creazione del codice. A partire da dicembre 2013 è stata rilasciata la versione 2 dell'ambiente, che va a sostituire la precedente versione beta (rinominata App Inventor Classic), introducendo una serie di migliorie e di fixing dei bug rispetto alla prima release.

Setup dell'ambiente e creazione di un Progetto

MIT App Inventor è un ambiente di sviluppo completamente on-line; per utilizzare l'ambiente bisogna dotarsi di un account Google e collegarsi all'indirizzo web <http://appinventor.mit.edu/explore/>. Nella pagina che si apre, bisogna poi cliccare sul pulsante "Create apps!", come illustrato in Fig. 3: verrà richiesto di selezionare un account Google e la relativa password ed una volta autenticati, anche di accettare le condizioni di utilizzo. Una volta accettate le condizioni, verrà aperta la pagina iniziale dell'ambiente di sviluppo on-line.

A questo punto è possibile selezionare il pulsante "Start New Project", presente in alto a sinistra, per creare un nuovo progetto. I file relativi ai progetti vengono mantenuti anch'essi su un database cloud e sono legati all'account Google utilizzato per l'autenticazione. Una volta selezionata l'opzione "Start New Project" si aprirà un pop-up all'interno del quale bisognerà inserire il nome del progetto; eseguita questa operazione il progetto comparirà tra l'elenco dei progetti ("My Projects", Fig. 4) e si aprirà la schermata di Fig. 5.

Debug ed Emulazione

Per poter testare e debuggare le applicazioni che andremo a creare ci occorre chiaramente un sistema



Fig. 3 - pagina iniziale di MIT App Inventor.



Fig. 4 - Nuovo progetto creato ed aggiunto alla lista "My Projects".

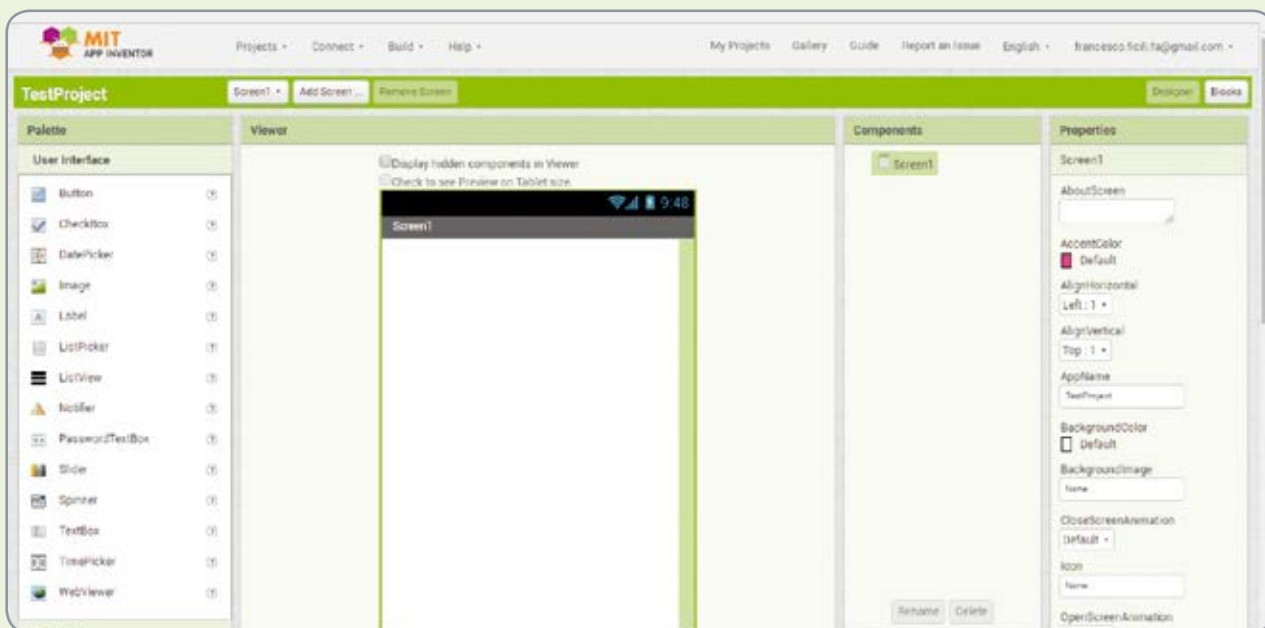


Fig. 5 - Schermata iniziale "TestProject".

per connetterci con un device di test, oppure un emulatore. App Inventor ci mette a disposizione ben tre opzioni per testare le nostre creazioni:

- test in real-time su device Android con connessione WiFi;
- test in real-time con Emulatore su PC,
- test in real-time su device Android con connessione USB.

In questa sezione descriveremo come effettuare il testing tramite le prime due opzioni (Fig. 6), in quanto consideriamo la terza semplicemente un sottocaso della prima, necessario solo in situazioni particolari nelle quali non sia disponibile la connessione WiFi sul nostro PC o sul device (caso peraltro estremamente raro). L'uso dell'emulatore è invece utile nel caso non si abbia a disposizione un dispositivo Android da utilizzare come device di test.

Test in real-time su device Android con WiFi

Occupiamoci della procedura per eseguire il test in

real-time tramite connessione WiFi:

1. installare sul dispositivo target l'app "MIT AI2 Companion", che può essere comodamente scaricata dal Play Store; una volta scaricata l'app è sufficiente seguire i vari step per l'installazione;
2. connettere il PC di sviluppo e il dispositivo target sulla stessa rete WiFi;
3. connettere il progetto App Inventor al dispositivo, selezionando l'opzione "Connect→AI Companion" presente sulla project bar in alto, come illustrato in Fig. 7.

Una volta selezionata questa opzione apparirà una finestra di dialogo contenente un QR code: a questo punto potete lanciare l'app sul dispositivo e connetterla al PC tramite una delle due opzioni disponibili (scan del QR code oppure inserimento diretto del codice a 6 cifre). Dopo qualche secondo avverrà la connessione e dovrete vedere sullo schermo del vostro dispositivo l'app che state sviluppando. L'app stessa si aggiornerà ogni volta che farete una

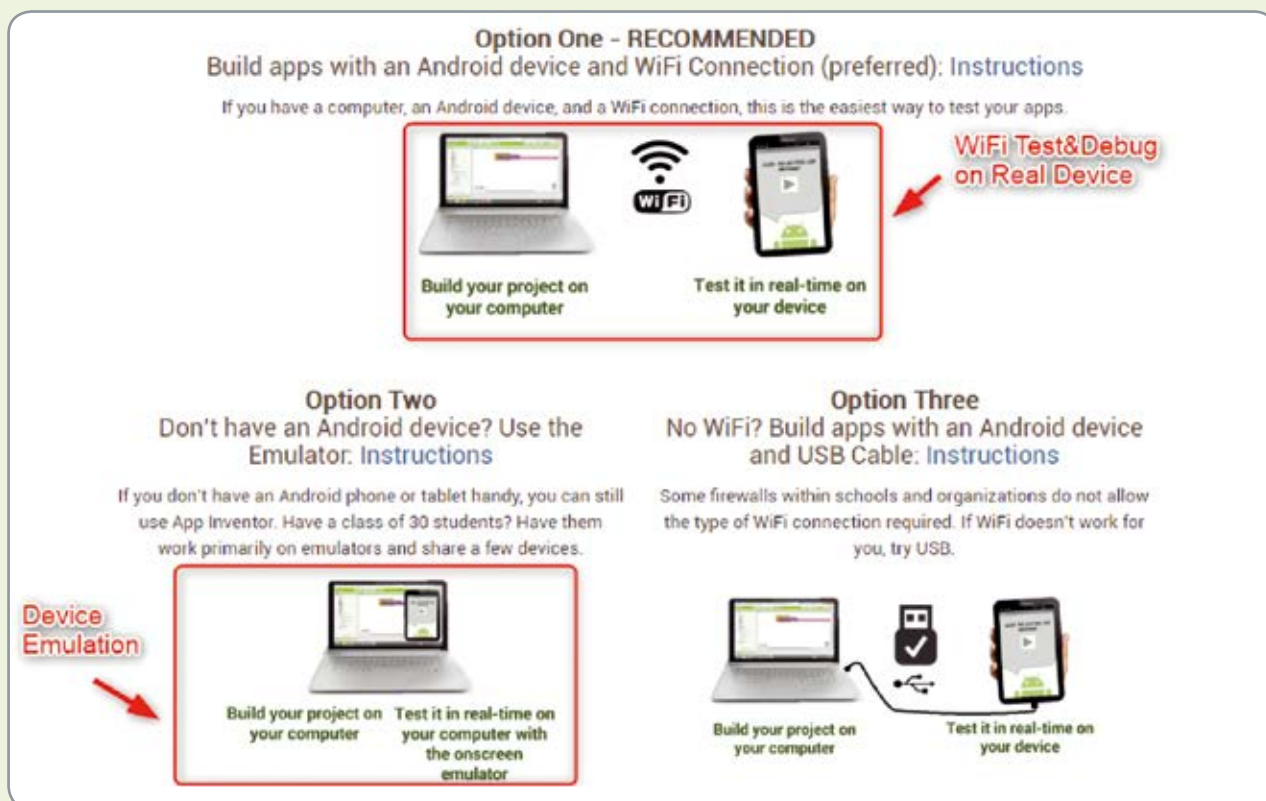


Fig. 6 - Possibili opzioni per il test e il debug delle app sviluppate con App Inventor.

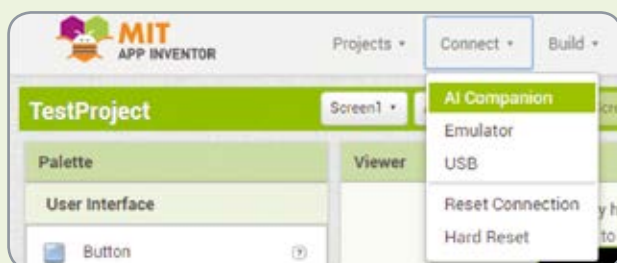


Fig. 7 - Connessione all'AI Companion.

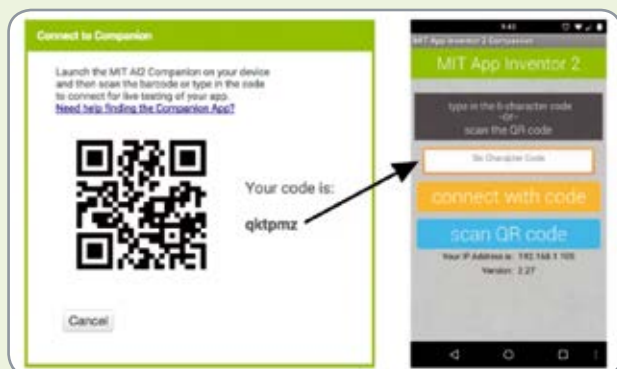


Fig. 8 - Scanning del QR code per connettere l'app.

modifica dall'ambiente di sviluppo, grazie ad una caratteristica chiamata "live testing".

Test in real-time con Emulatore

L'utilizzo dell'emulatore permette di testare le nostre app senza l'utilizzo di un device esterno Android, ma avvalendosi di un dispositivo virtuale che gira direttamente sul PC di sviluppo. È un'opzione molto utile in diverse circostanze e ce ne avvarremo per comodità didattica durante il corso, tutte le volte che sarà possibile.

Per poter utilizzare l'emulatore bisogna prima di tutto installarlo. Per farlo è necessario:

1. scaricare l'applicazione (questo link è valido per la versione Windows...) da <http://appinventor.mit.edu/explore/ai2/windows.html>; per le istruzioni complete per MAC OS e Linux vi rimandiamo all'apposita sezione sul sito di App Inventor (<http://appinventor.mit.edu/explore/ai2/setup-emulator.html#step2>).
2. una volta installato l'emulatore bisogna lanciarlo con un doppio clic sull'icona **ai Starter** che troverete sul desktop; verrà aperta una command window che visualizzerà sulla console i messaggi provenienti dall'aiStarter;

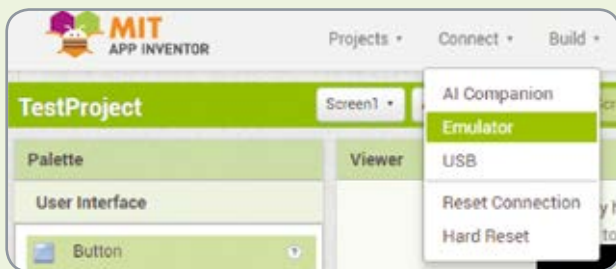


Fig. 9 - Connessione dell'emulatore dall'ambiente di sviluppo.

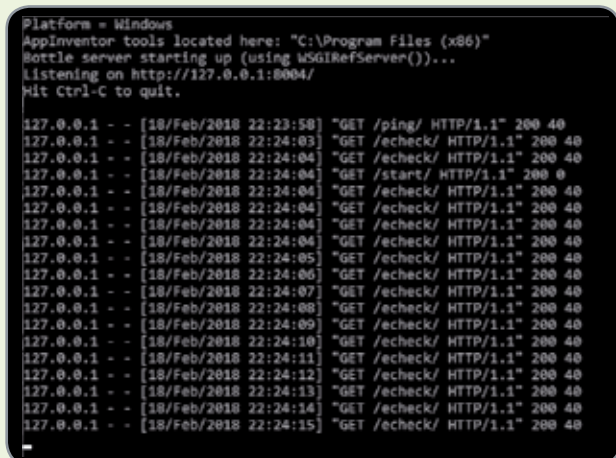


Fig. 10 - Connessione dell'emulatore, messaggi sulla console.

3. adesso è sufficiente selezionare l'opzione "Connect→Emulator" presente sulla project bar in alto, come illustrato in Fig. 9.

A questo punto dovreste notare alcune attività in svolgimento sul lato aiStarter, come illustrato in Fig. 10, e dopo alcuni secondi verrà avviato anche l'emulatore del dispositivo; infine verrà sincronizzata l'app (Fig. 11) realizzata.

Da qui in poi il funzionamento è del tutto simile al caso del dispositivo esterno, a meno di limitazioni nell'uso delle periferiche.

Designer view e Block view

Passiamo ora all'analisi dell'ambiente di sviluppo ed in particolare soffermiamoci sulle due schermate principali nelle quali ci troveremo a lavorare; l'interfaccia di App Inventor è infatti costituita da due schermate principali di lavoro, chiamate **Designed View** e **Block View**: nella prima si sviluppa il layout della nostra app, ossia si costruisce la sua interfaccia grafica, mentre nella seconda si inserisce il codice vero e proprio, che, come abbiamo visto in

precedenza, in App Inventor è totalmente grafico. Vediamo quindi come sono composte queste due View, partendo dalla Designer View ed aiutandoci con la Fig. 12: nella Designer View troviamo i seguenti componenti principali:

- **Main Bar** che è la barra principale di gestione del progetto, dalla quale è possibile gestire gli screen (aggiungere nuove viste o eliminare viste esistenti), effettuare tutte le operazioni di project management, eseguire il build di una app ed altro ancora;
- **Components Palette**, che è la palette contenente i componenti e che, come di consueto, è divisa per categorie; i componenti sono una caratteristica molto importante di App Inventor, in quanto oltre a permettere la costruzione dell'interfaccia grafica e del layout, permettono di gestire praticamente tutte le periferiche presenti sul nostro Smartphone; consentono anche di eseguire operazioni avanzate, come ad esempio l'interfacciamento con i social, la gestione dello storage e della connettività, ecc.
- **Viewer**: questa sezione rappresenta la nostra area di lavoro in fase di layout design, infatti è sagomata come lo schermo del nostro Smartphone; in



Fig. 11 - Sincronizzazione dell'app sul dispositivo emulato.

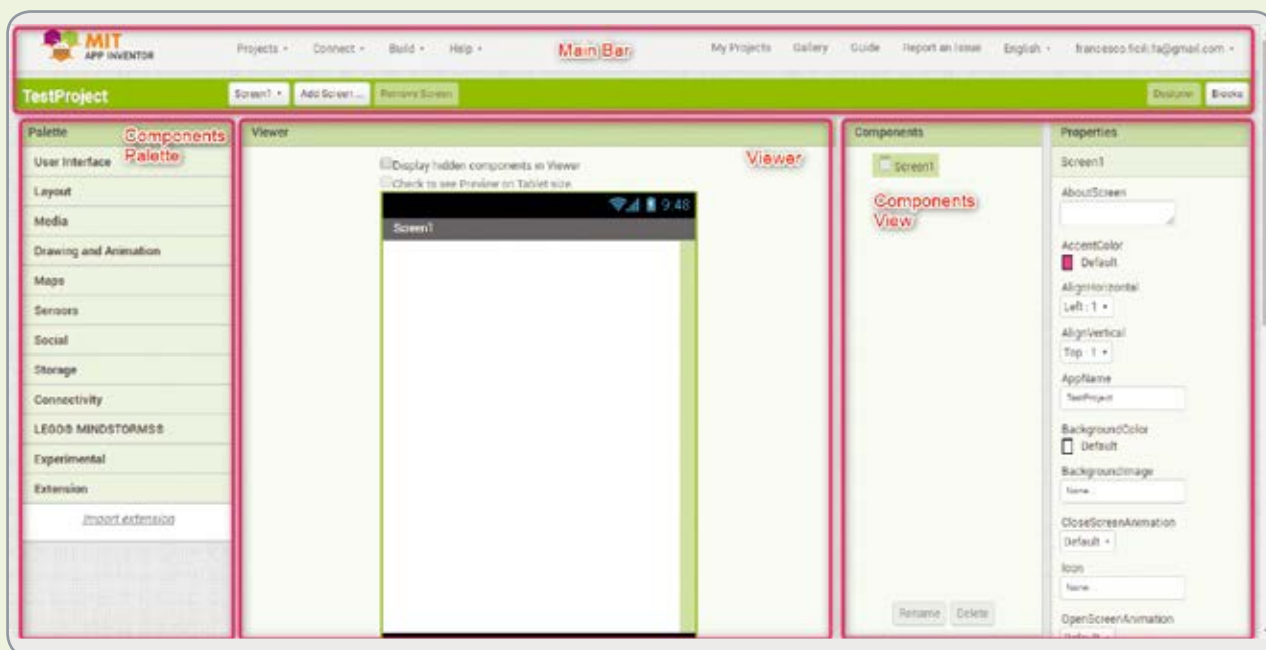


Fig. 12 - La Designer View.



Fig. 13 - Selezione della Block View.

essa è possibile trascinare i componenti prelevati dalla Components Palette.

- **Components View**, dalla quale è possibile gestire le proprietà dei componenti, oltre che rinominare o eliminare gli stessi dal layout; questa view è divisa in due sezioni chiamate **Components** (da cui si può selezionare uno dei componenti presenti all'interno di un determinato screen) e **Properties**, dalla quale è possibile accedere le proprietà del componente selezionato.

Dalla Designer View è possibile passare alla Block View tramite il selettore presente alla estrema destra della Main Bar, selezionando la voce Blocks, come illustrato in Fig. 13. La Block View, visibile in Fig. 14, è la schermata all'interno della quale avviene lo sviluppo della logica della nostra app; essa è composta dai seguenti elementi:

- **Main Bar**, in tutto e per tutto identica alla Main Bar esaminata nel caso della Designer View;

- **Blocks Palette**, la quale contiene i blocchi per l'implementazione della nostra applicazione; tale palette è divisa in due parti: la prima (denominata Built-in) contiene tutti i blocchi base, come i blocchi logico/matematici, i blocchi per il controllo di esecuzione, le varie costanti e le varie operazioni di base che sono eseguibili sui blocchi, mentre la seconda contiene i blocchi specifici dei vari componenti del progetto e tale sezione si popola man mano che i componenti vengono aggiunti alla View lato design;
- **Blocks Viewer**: questa è la sezione nella quale viene creato il codice grafico, trascinando all'interno di essa i vari blocchi presenti della Blocks Palette; per impedire errori nelle connessioni i blocchi in App Inventor sono sagomati a forma di puzzle e si possono incastrare tra di loro solo se risultano compatibili.

Esempio Grafico: Hello World

Ora che abbiamo preso un po' di confidenza con l'ambiente di sviluppo, passiamo alla realizzazione di un semplice esempio pratico che ci permetta di cogliere subito la potenza di questo strumento di programmazione: costruiamo il classico programma "Hello World", ma anziché far lampeggiare un LED o stampare la classica stringa "Hello World" a video, utilizziamo il TextToSpeech component per sintetizzare vocalmente la nostra stringa "Hello World".

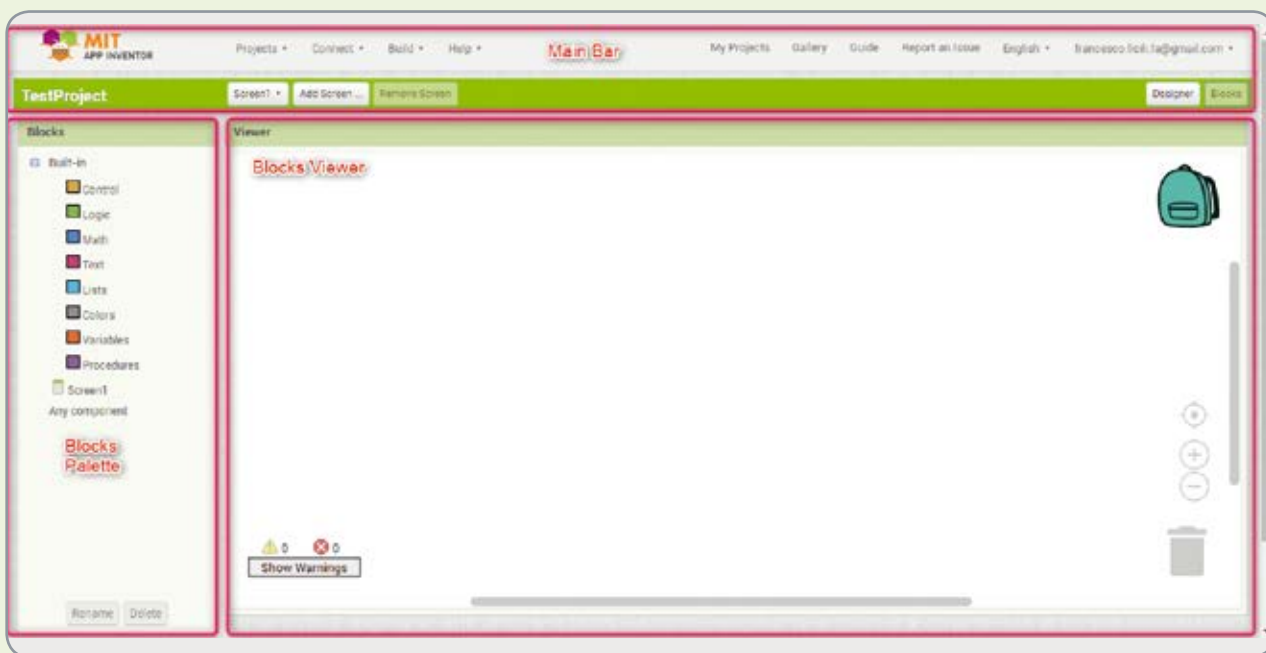


Fig. 14 - Block View.

Partiamo dalla Designer View ed inseriamo sul nostro layout i seguenti componenti:

- User Interface → Button,
- Media → TextToSpeech.

Come già accennato, per inserire i componenti sul layout occorre selezionarli dalla palette e poi tra-

scinarli sulla Design Viewer. Una volta completata questa operazione, il nostro layout dovrebbe essere del tutto simile a quello riportato in Fig. 15. A differenza del button il componente TextToSpeech è un componente non visibile, quindi la sua presenza all'interno del progetto viene indicata tramite una icona in basso sotto il layout.

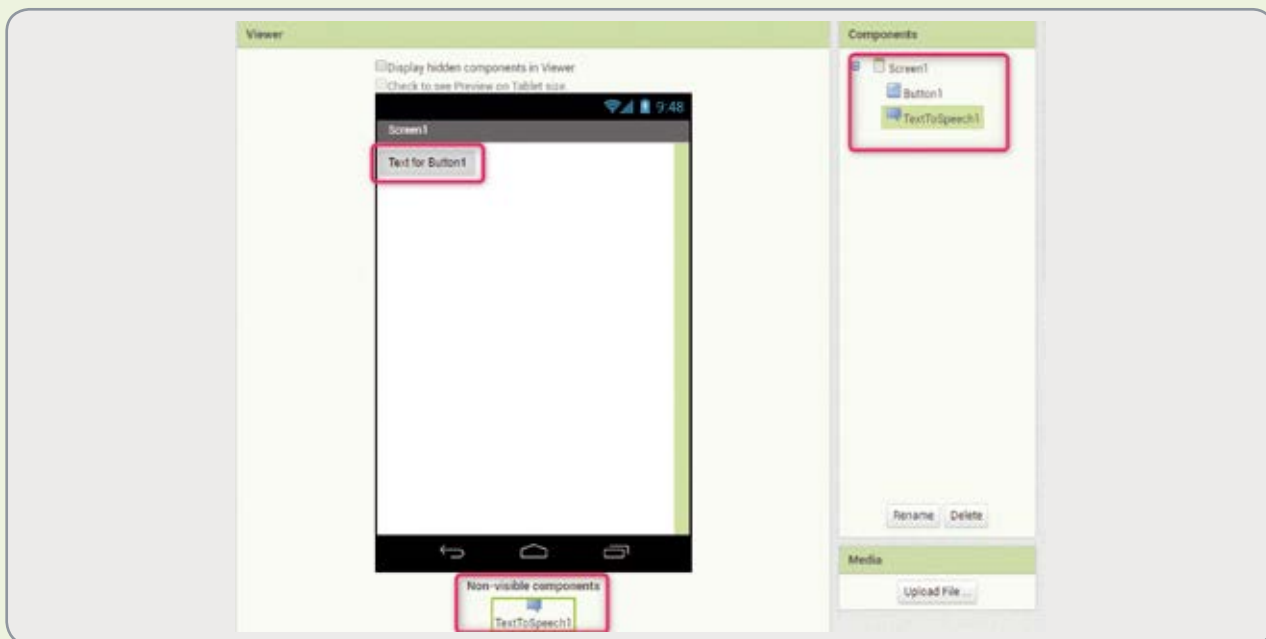


Fig. 15 - Inserimento dei componenti sulla Design View.

Breve storia di Android

Nell'ottobre del 2003 quattro programmatori statunitensi, Andy Rubin, Chris White, Nick Sears e Rich Miner fondarono una società, la Android Inc. per lo sviluppo di quello che Rubin definì "...dispositivi cellulari più consapevoli della posizione e delle preferenze del loro proprietario". Durante lo stesso anno il budget inizialmente stanziato per il progetto si esaurì ed un amico di Rubin, Steve Perlman, rifinanziò il progetto con 10.000 dollari. Perlman consegnò a Rubin il denaro in una busta ma rifiutò ogni proposta di partecipazione alla società. Il 17 agosto 2005 Google acquistò l'azienda, come acquisizione strategica per l'ingresso nel mercato della telefonia mobile. È in quel periodo che il team comincia a sviluppare un sistema operativo per dispositivi mobili basato su Linux. La presentazione ufficiale dell'OS avvenne il 5 novembre 2007 dalla neonata OHA (Open Handset Alliance), un consorzio di aziende del settore Hi Tech che include Google, produttori di smartphone, operatori di telefonia mobile e produttori di microprocessori come Qualcomm e TI. Il primo dispositivo equipaggiato con Android che venne lanciato sul mercato fu l'HTC Dream: era il 22 ottobre del 2008. Dalla versione 1.5 di Android, ogni aggiornamento o rilascio dell'OS, segue una preci-

sa convenzione alfabetica per i nomi, associando ogni release al nome di un dolce:

- 1.5 ————— Cupcake,
- 1.6 ————— Donut,
- 2.0, 2.1 ————— Eclair,
- 2.2.x ————— Froyo,
- 2.3.x ————— Gingerbread,
- 3.0, 3.1, 3.2.x ——— Honeycomb,
- 4.0.x ————— Ice Cream Sandwich,
- 4.1.x, 4.2.x, 4.3.x — Jelly Bean,
- 4.4.x ————— KitKat
- 5.0.x, 5.1.x ————— Lollipop,
- 6.0.x ————— Marshmallow,
- 7.0, 7.1.x ————— Nougat,
- 8.0, 8.1 ————— Oreo



Fig. A - Logo di Android Oreo.

Per mantenere un minimo di ordine e coding style rinominiamo i componenti come segue:

- Button1 → Btn_HelloWorld;
- TextToSpeech1 → Cmp_TextToSpeech.

I componenti possono essere rinominati selezionandoli nella component view e cliccando sul tasto Rename, presente in basso sulla stessa view. Comparirà una finestra di dialogo dalla quale sarà possibile rinominare il componente. Inoltre modifichiamo la proprietà Text del Button, cambiandola in "Hello World!!!", come indicato in Fig. 16. Poi settiamo la lingua predefinita per il componente TextToSpeech su "Italiano", attraverso il menu a tendina Language accessibile dalle proprietà del componente.

A questo punto possiamo passare nella Block View

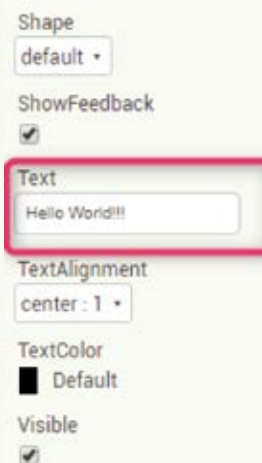


Fig. 16 - Aggiornamento proprietà Text del Button.



Fig. 17 - Blocco When Btn_HelloWorld.Click...do.



Fig. 18 - Blocco call Cmp_TextToSpeech.Speak.

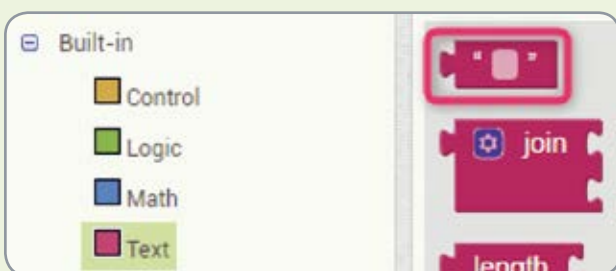


Fig. 19 - Blocco Text String.

per aggiungere il codice grafico alla nostra app. Procediamo come segue: per prima cosa posizioniamo sulla Viewer il blocco "When Btn_HelloWorld.

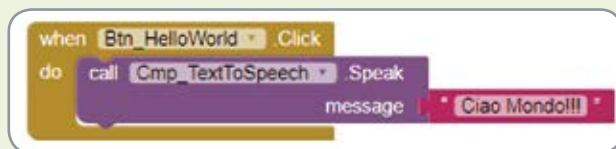


Fig. 20 - Codice grafico relativo all'esempio pratico.

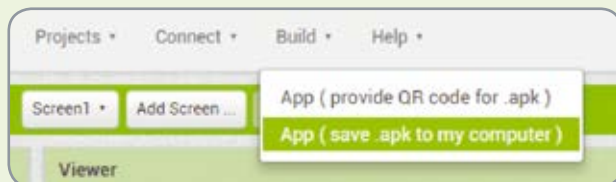


Fig. 21 - Building dell'App definitiva.

Click...do", presente tra i blocchi specifici del componente Btn_HelloWorld (Fig. 17). Poi incastriamo all'interno di quest'ultimo il blocco "call Cmp_TextToSpeech.Speak" del componente Cmp_TextToSpeech (Fig. 18). Infine colleghiamo alla connessione "message" di quest'ultimo un blocco text string (Fig. 19).

Infine scriviamo all'interno del blocco text string la stringa "Ciao Mondo!!!". Se avete fatto tutto correttamente il risultato dovrebbe essere visibile in Fig. 20.

Per testare la nostra app così creata possiamo utilizzare uno dei due metodi descritti precedentemente (test su dispositivo tramite connessione WiFi o uso dell'emulatore). Una volta sicuri del risultato possiamo anche eseguire il build della nostra app in formato .apk ed installarla sul nostro dispositivo mobile. Per farlo è sufficiente selezionare l'opzione Build→App (save .apk on my computer) presente nella Main Bar, come illustrato in Fig. 21.

Completata la fase di Build, l'app verrà scaricata nella cartella predefinita per i download del browser che state utilizzando e potrete installarla sul vostro dispositivo mobile.

Conclusioni

Siamo arrivati alla conclusione di questa prima puntata di questo corso introduttivo a MIT App Inventor; in questa puntata abbiamo iniziato a prendere confidenza con l'ambiente, installato l'emulatore, analizzato l'interfaccia e creato un primo progetto di esempio. Dalla prossima puntata inizieremo con i primi esempi pratici di applicazioni Android, occupandoci dei concetti di base relativi alla programmazione grafica e dando un primo sguardo ai componenti, che sono uno degli elementi fondamentali di App Inventor.



MIT APP INVENTOR

2

di FRANCESCO FICILI

Continuiamo il nostro viaggio alla scoperta di MIT App Inventor, entrando nel merito della programmazione grafica e illustrando alcune delle proprietà più interessanti di questo eccezionale tool per lo sviluppo di app Android. Seconda Puntata.

Nella prima puntata di questo corso abbiamo introdotto MIT App Inventor (abbreviato di seguito con AI), l'innovativo ambiente di sviluppo grafico per applicazioni Android creato da Google e sviluppato attualmente dal MIT. Abbiamo presentato l'ambiente e l'emulatore, approfittando per presentare un primo, semplice, esempio. In questa puntata entriamo maggiormente nel merito della programmazione grafica, illustrando alcune delle proprietà più interessanti di AI.

Programmazione in MIT App Inventor

Sapete ormai che in AI la programmazione avviene completamente tramite l'uso di elementi grafici.



Fig. 1 - Insieme dei blocchi built-in.

Fig. 2 - Esempio di blocchi del gruppo control.

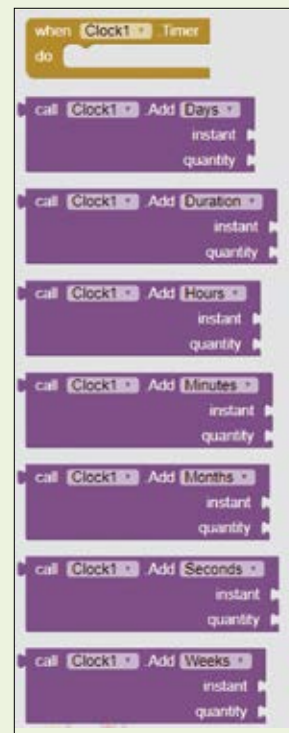


Fig. 3 - Esempio di blocchi del componente clock.

Tali elementi in AI sono generalmente chiamati “Behaviours” (comportamenti) e prendono la forma di blocchi simili alle tessere di un puzzle (nel resto della puntata e nelle successive utilizzeremo il termine blocchi e behaviours indistintamente, riferendoci sempre a questi elementi). I blocchi in AI possono appartenere fondamentalmente a due gruppi:

- built-in; si tratta dei blocchi base, sempre presenti a prescindere dai componenti inseriti in una determinata app (in Fig. 1 è riportato l'insieme dei blocchi built-in e in Fig. 2 alcuni blocchi del gruppo control);
- componenti; si tratta dei blocchi specifici per i componenti; ne esiste una grande varietà, in quando questi blocchi cambiano da componente a componente, ma si possono identificare delle classificazioni (in Fig. 3 è riportato un esempio per il componente clock).

Beige	Strutture di controllo
Verde	Blocchi logici
Blu	Blocchi matematici
Fucsia	Blocchi di testo (stringhe)
Celeste	Blocchi gestione Liste
Grigio	Blocchi gestione colori
Arancio	Blocchi gestione variabili
Viola	Blocchi controllo procedure

Tabella 1 - Significato dei colori per i componenti built-in.

Beige	Eventi
Fucsia	Metodi
Verde Scuro	Setter di proprietà
Verde Chiaro	Getter di proprietà

Tabella 2 - Significato dei colori per i behaviours più comuni dei componenti.

In App Inventor, forma e colore di un behaviour hanno un loro significato; in particolare si usa, per i blocchi built-in, l'insieme di associazioni riportato in **Tabella 1**, mentre per quanto riguarda i behaviour dei componenti, i più comuni significati dei colori sono riportati in **Tabella 2**.

Inoltre, sostanzialmente tutti i blocchi presentano uno o più connettori di ingresso e di uscita, sagomati come rientranze o sporgenze, tipo tasselli di puzzle. In prossimità del connettore è solitamente presente una label che serve ad identificare l'input o l'output del connettore stesso.

Abbiamo iniziato ad anticipare dei concetti parlando di eventi, metodi, setter e getter, che saranno chiariti nei paragrafi successivi, quando parleremo dei componenti. Prima di arrivare a quel punto però, occorre illustrare come AI gestisce due aspetti che sono alla base di qualsiasi linguaggio di programmazione: le strutture dati e le strutture di controllo.

Variabili

Uno degli aspetti fondamentali di un qualsiasi linguaggio di programmazione è la gestione delle strutture dati, ossia le tipologie di dati supportate e le operazioni che possono essere eseguite su di esse. AI prevede una gestione estremamente semplificata delle strutture dati: si possono dichiarare variabili

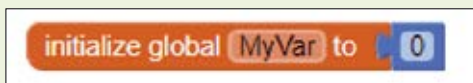


Fig. 4 - Creazione e inizializzazione di una variabile globale.



Fig. 5 - Blocchi di inizializzazione di variabili local in AI.

globali e locali, mentre sono supportati i seguenti tipi di dati:

- numerici;
- booleani;
- stringhe;
- liste;
- colori.

Vediamo come definire variabili globali e locali tramite un semplice esempio.

Per definire una variabile globale è sufficiente posizionare il blocco "initialize global name to" del gruppo built-in "Variables" in un punto qualsiasi del nostro Block Viewer e rinominare la sezione "name" con il nome della nostra variabile (nel nostro esempio è *MyVar*), come illustrato in Fig. 4. In

questo caso viene creata una variabile globale denominata *MyVar* ed inizializzata al valore numerico '0'. Tale variabile sarà visibile ovunque all'interno della nostra app.

Le variabili locali invece hanno una visibilità (o *scope*) limitato ad una specifica procedura; tale tipologia di variabile può essere definito tramite due blocchi distinti in AI, come illustrato in Fig. 5.

La differenza sta solo nel fatto che il primo blocco può essere utilizzato nella sezione **do** di un blocco superiore (tipicamente un evento), mentre il secondo nella sezione **return** di un blocco collegato: ad esempio un blocco di set (come abbiamo visto in precedenza i blocchi in AI sono sagomati in modo da incastrarsi tra loro solo in un determinato modo, così da garantire intrinsecamente la compatibilità). In Fig. 6 è illustrato un esempio che mostra l'utilizzo dei due blocchi per settare il valore di una label; nell'esempio vengono utilizzati due eventi (nello specifico *Screen1.Initialize* e *Screen1.ScreenOrientationChanged*) per settare il valore della label *Label1* al valore corrispondente, rispettivamente, alle variabili locali *MyLocal1* e *MyLocal2*, recuperati tramite l'uso di un blocco "get".

Nell'esempio appena visto abbiamo introdotto anche una delle due operazioni fondamentali che possono essere eseguite su una variabile, ossia l'operazione di **get** e quella, altrettanto importante, di **set**, che si utilizza quando vogliamo assegnare un valore ad una variabile. È importante notare che ogni volta che viene creata una variabile, locale o globale, se ci posizioniamo con il puntatore del mouse sopra il nome della variabile stessa, compare sullo schermo una shortcut che ci permette di accedere velocemente ai blocchi di get e set per quella variabile, come si vede in Fig. 7.

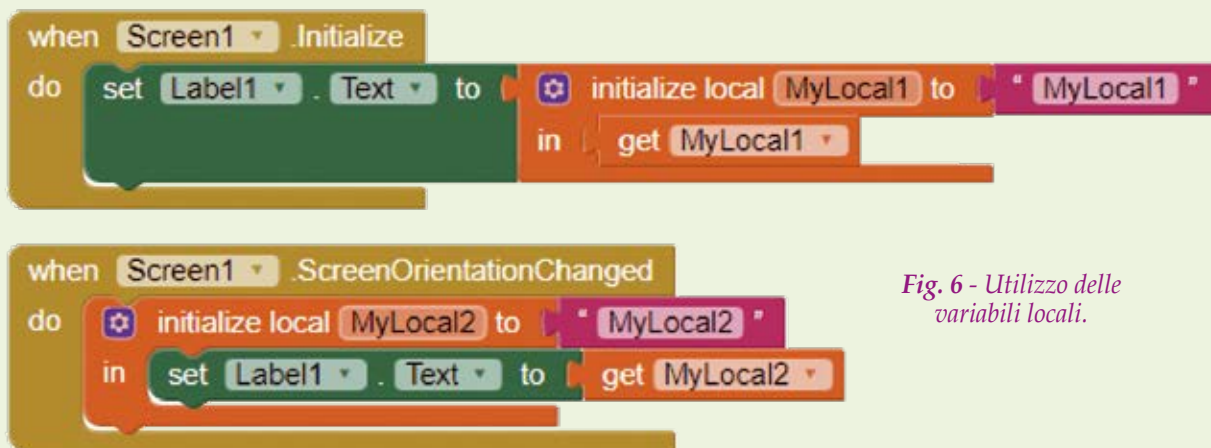


Fig. 6 - Utilizzo delle variabili locali.

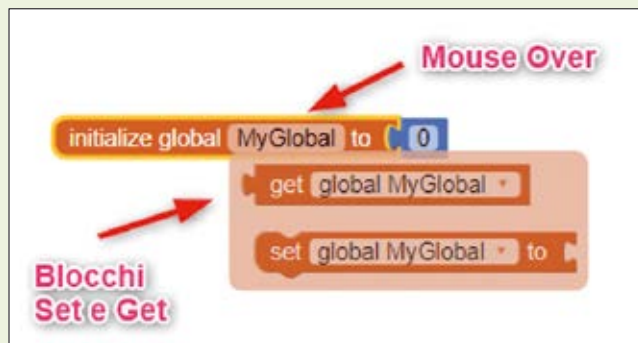


Fig. 7 - Blocchi Set e Get per la variabile MyGlobal.

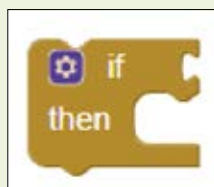


Fig. 8
Blocco if..then base.

Strutture di controllo

Passiamo ora all'analisi dell'altro aspetto fondamentale del linguaggio dopo le strutture dati, ossia le strutture di controllo. AI presenta le strutture di controllo classiche che possono essere trovate nei più comuni linguaggi, più ulteriori strutture espressamente pensate per la programmazione di applicazioni in ambiente android. AI racchiude tutte queste strutture all'interno del gruppo di behaviours built-in denominato "control"; essenzialmente si possono identificare tre gruppi di strutture di controllo:

- strutture di controllo del flusso di esecuzione;
- loop;
- strutture di controllo dell'applicazione.

Controllo di Flusso

Le strutture di controllo di flusso consentono di eseguire operazioni di tipo decisionale, per mezzo di blocchi condizionali come i blocchi "if..then" e "if..then..else". Tali blocchi operano su risultati di operazioni booleane o di confronto. Il blocco base è il blocco "if...then", rappresentato nella sua forma base in Fig. 8, che data una condizione iniziale, da collegare sull'ingresso "if" la verifica e, in caso sia vera esegue il ramo "then", altrimenti la salta.

Tale blocco può essere modificato cliccando sul pulsante a forma di ingranaggio, aggiungendo ulteriori branch, come illustrato in Fig. 9, dove un blocco if...then base viene modificato in una condizione doppia if..then..else if. Rispetto al caso precedente, in questo frangente viene valutata la condizione relativa all'ingresso "if", se quest'ultima è vera, viene eseguito il ramo "then", altrimenti viene saltata e si passa alla verifica della condizione del ramo "else if". Su un blocco if..then può essere inserito un numero arbitrario di rami "else..if", ma solo un ramo "else", che identifica il ramo da eseguire nel caso nessuna delle condizioni precedenti sia stata verificata.

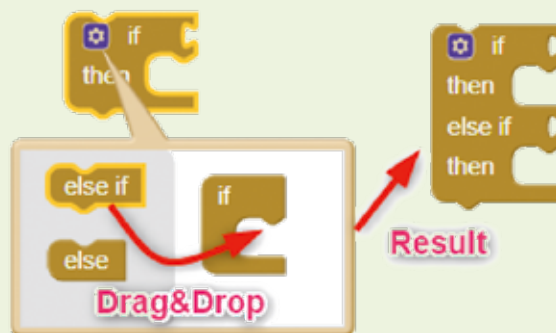


Fig. 9 - Modifica di un blocco if..then..else if.

Loop

I loop (o cicli) sono blocchi pensati per l'esecuzione di operazioni iterative, sia a conteggio (cicli for) che a condizione d'uscita (cicli while). Partiamo dai cicli a conteggio: AI mette a disposizione due tipi di ciclo "for": uno generico ed uno specifico per la gestione delle liste, entrambi illustrati in Fig. 10.

Il primo è il classico ciclo a conteggio, che definisce al suo interno una variabile locale number ed esegue il blocco di codice collegato sul connettore "do" per valori della variabile number compresi tra gli estremi "from" e "to", incrementando a ogni iterazione number di una quantità pari a "by". I valori di "from", "to" e "by" possono essere costanti oppure calcolati tramite una opportuna espressione matematica. Invece il blocco "for each item in list" esegue semplicemente un numero di iterazioni pari al numero di elementi presenti nella lista che viene collegata al suo connettore "list". Per quanto riguarda invece i cicli a condizione di uscita AI rende disponibile un unico ciclo "while", rappresentato in Fig. 11.

Il funzionamento è semplicissimo, il blocco valuta ad ogni iterazione la condizione collegata al terminale "test", se questa è verificata esegue il codice grafico collegato al connettore "do", altrimenti termina. Chiaramente, anche in questo caso la condizione può essere una costante o un'espressione complessa a piacere; l'importante è che il risultato alla fine sia un valore booleano.

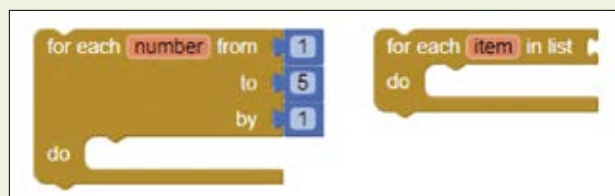


Fig. 10 - Tipologie di cicli for supportate da AI.



Fig. 11 - Ciclo While.

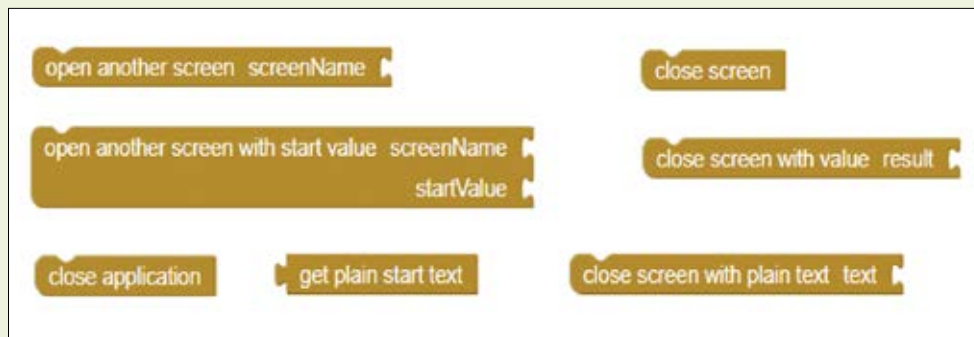


Fig. 12 - Strutture di controllo dell'applicazione.

Controllo dell'applicazione

Veniamo infine ai blocchi di controllo dell'applicazione; AI mette a disposizione diverse strutture di controllo di questo tipo, in Fig. 12 ne abbiamo riportate alcune.

Tali strutture di controllo permettono di gestire ad alto livello la nostra app, ad esempio ci consentono di passare da uno screen ad un altro in app multi-screen, chiudere o aprire screen e chiudere l'app stessa.

I componenti in App Inventor

Ora che abbiamo illustrato strutture dati e strutture di controllo, possiamo spiegare cosa si intende in AI per componenti. I componenti (o components) non sono altro che gli elementi che ci consentono di aggiungere funzionalità alla nostra app. Come abbiamo visto nella prima puntata, possiamo inserirli nella "Designer View" trascinandoli dalla "Components Palette" all'interno della "Designer View", dove è rappresentato il layout della nostra app. I componenti possono essere visibili o meno, a seconda del fatto che abbiamo o meno una componente grafica; ad esempio un componente button è visibile sul layout in quanto ha sia una componente grafica (l'immagine del pulsante stesso) che una serie di blocchi associati. Un componente Accelerometer, invece, è non visibile, in quanto non ha parte grafica ma solo una serie di blocchi associati al funzionamento nella "Block View".

I componenti non visibili vengono comunque visualizzati come icone nella parte bassa della "Designer View", in modo da poter essere comunque selezionati e da poter accedere alle loro proprietà. Prima di proseguire facciamo una veloce rassegna dei componenti principali di App Inventor.

User Interface: si tratta della famiglia di componenti più estesa, tramite la quale è possibile creare l'interfaccia grafica della nostra app. A questa famiglia appartengono i seguenti elementi:

- Button; il classico pulsante grafico;
- CheckBox; box di scelta true/false;
- DatePicker; picker che permette di selezionare una data tramite finestra di pop-up;
- Image; permette di mostrare un'immagine sul layout della nostra app;
- Label; consente di stampare del testo a video;
- ListPicker; picker che permette di selezionare un elemento da una lista;
- ListView; permette di visualizzare gli elementi di una lista;
- Notifier; permette di visualizzare dei messaggi all'utente;
- PasswordTextBox; elemento che permette di inserire un input testuale in modalità password (invece dei caratteri alfanumerici vengono stampati degli asterischi);
- Screen; serve a inserire schermate aggiuntive;
- Slider; elemento che permette all'utente di fornire un input numerico muovendo il cursore di una slide bar;
- Spinner; elemento che permette di creare un drop-down menu;
- TextBox; elemento che permette di inserire un input testuale;
- TimePicker; picker che permette di selezionare un'ora tramite finestra di pop-up;
- WebView; elemento che permette di aprire e visualizzare una URL.

Layout: si tratta di una famiglia di componenti che permettono di organizzare il layout della nostra app, creando una sorta di griglia che permette di collocare gli elementi grafici sul layout. Ci sono sostanzialmente tre possibili scelte per il layout:

- VerticalArrangement; è layout verticale;
- HorizontalArrangement; layout orizzontale;
- TableArrangement; è il layout tabellare.

Media: questa famiglia di componenti consente di gestire elementi multimediali all'interno della nostra app. All'interno di questa famiglia troviamo:

- Camcorder; è il componente che permette di aprire la videocamera del dispositivo per effettuare la registrazione di un video;
- Camera; permette di aprire la videocamera del dispositivo per scattare una foto;
- ImagePicker; è il componente che consente di selezionare un'immagine tra quelle presenti nella galleria del device;
- Player; permette di riprodurre un file audio e di controllare la vibrazione del device (per file audio di lunga durata);
- Sound; consente di riprodurre un file audio e di controllare la vibrazione del device (per file audio di breve durata);
- SoundRecorder; permette di accedere al microfono del dispositivo ed effettuare una registrazione audio;
- SpeechRecognizer; permette di attivare la funzionalità di riconoscimento vocale integrata in Android al fine di convertire un parlato in testo;
- TextToSpeech; permette di trasformare un testo in un parlato attraverso un sintetizzatore vocale (supporto multilingua);
- VideoPlayer; permette di riprodurre un file video all'interno di un player dotato dei normali comandi attivabili tramite touch screen;
- YandexTranslate; permette di effettuare traduzioni in tempo reale attraverso le API offerte dal traduttore automatico di Yandex.

Drawing & Animation: questa famiglia offre una serie di componenti per gestire animazioni grafiche e disegni. L'elemento base è il "canvas", ossia una sorta di blocco all'interno del quale è possibile effettuare disegni ed animazioni. Questa famiglia comprende i seguenti componenti:

- Canvas; è un blocco rettangolare all'interno del quale è possibile disegnare o effettuare delle animazioni;
- Ball; sprite circolare che può essere posizionato all'interno di un canvas, dove può interagire con altri sprites e reagire al touch;
- ImageSprite; sprite che può essere associato ad un'immagine e posizionato all'interno di un canvas, dove può interagire con altri sprite e reagire al touch.

Maps: famiglia di component specifici per la gestione di mappe. All'interno di questa famiglia troviamo:

- Circle; è il componente che permette di visualizzare un cerchio di un certo raggio (in metri) ad una determinata latitudine e longitudine;
- FeatureCollection; fornisce un contenitore per caricare caratteristiche multiple quali gruppo e

gruppo di poligoni rappresentanti gli stati degli United States (ciascuna caratteristica diviene un proprio componente che può essere editato nel designer);

- LineString; componente che permette di disegnare una sequenza di linee su una mappa;
- Map; componente che esegue il rendering di mappe e permette di posizionare marker per identificare punti sulla mappa renderizzata;
- Marker; componente che permette di posizionare un marker su una mappa;
- Polygon; componente che permette di disegnare un poligono su una mappa;
- Rectangle; è il componente che permette di disegnare un rettangolo su una mappa.

Sensors: famiglia di componenti che permette di accedere ai sensori interni del dispositivo. Tra essi troviamo:

- Accelerometer; componente che permette di interfacciare l'accelerometro interno;
- BarcodeScanner; componente che permette di utilizzare la camera come scanner per codici a barre;
- Clock; componente che permette di generare eventi periodici;
- Gyroscope; componente che permette di interfacciare il giroscopio interno;
- LocationSensor; componente che permette di effettuare operazioni di geolocalizzazione;
- NearField; componente che permette di gestire l'interfaccia NFC del dispositivo;
- OrientationSensor; componente che permette di interfacciare la bussola elettronica interna;
- Pedomter; componente che consente, tramite l'accelerometro interno, di contare i passi;
- Proximity; è il componente che permette di rilevare la distanza di un oggetto (in cm) rispetto allo schermo.

Social: famiglia di componenti specifica per la gestione di funzionalità social. I componenti che fanno parte di questa famiglia sono:

- ContactPicker; picker che permette di selezionare un contatto dalla rubrica;
- EmailPicker; picker che permette di trovare un indirizzo email inserendo il nome del contatto;
- PhoneCall; componente che consente di effettuare una chiamata telefonica al numero specificato;
- PhoneNumberPicker; picker che permette di ricavare il numero di telefono inserendo il nome del contatto;
- Sharing; componente che permette di effettuare sharing (di file o messaggi) tra la app ed altre app installate sul dispositivo;

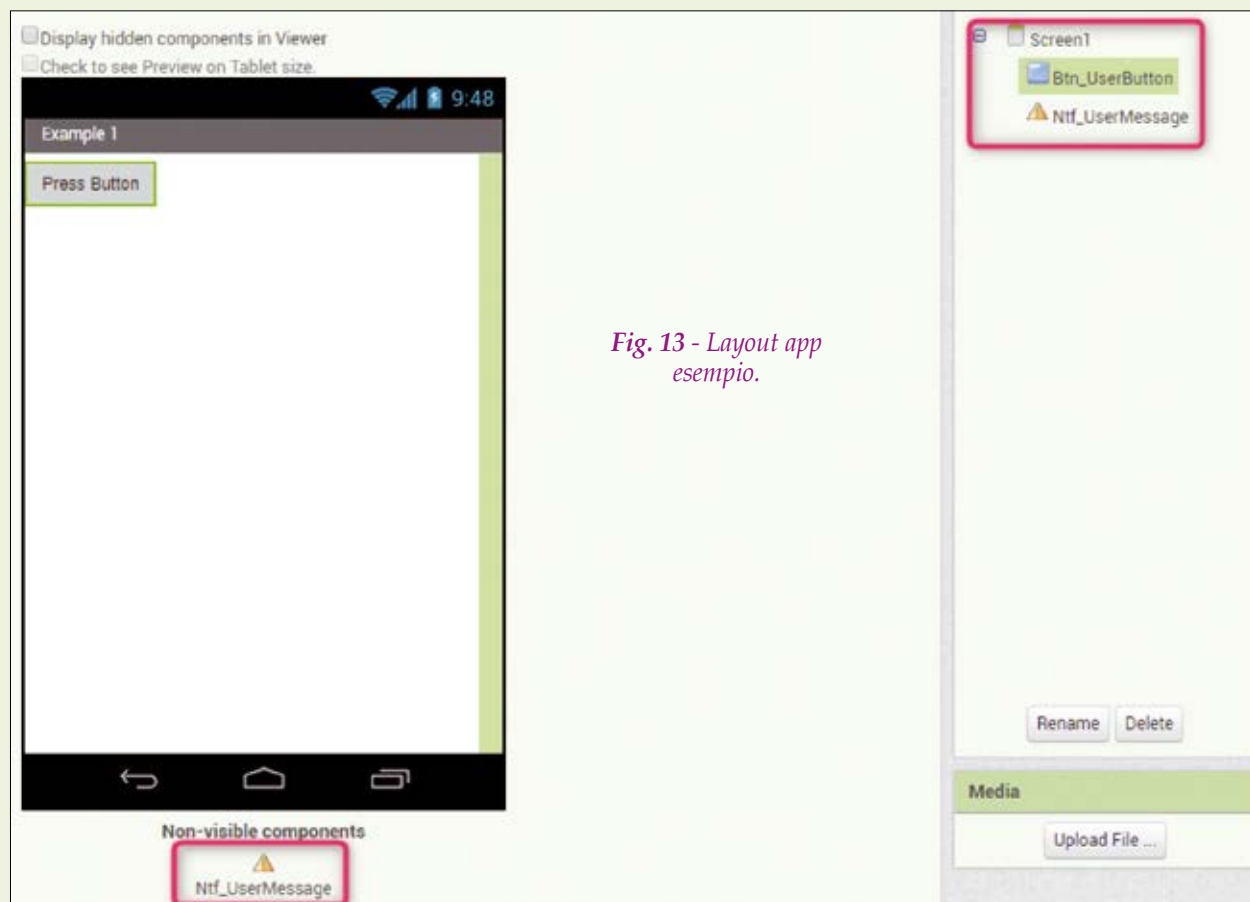


Fig. 13 - Layout app esempio.

- Texting; componente che permette di inviare e ricevere SMS;
- Twitter; componente che permette di interagire con twitter.

Storage: famiglia di componenti che permette la gestione della memorizzazione dei dati all'interno del dispositivo. Questa famiglia comprende i seguenti componenti:

- File; componente che permette di leggere e scrivere file;
- TinyDB; componente che permette di gestire un piccolo DB sul dispositivo;
- TinyWeDB; componente che permette di accedere ad un web service per immagazzinare e recuperare informazioni.

Connectivity: famiglia di componenti per la gestione della connettività. Questa famiglia comprende i seguenti componenti:

- ActivityStarter; componente che permette di lanciare una activity;
- BTClient; il componente che permette di gestire un client Bluetooth;
- BTServer; componente che permette di gestire un server Bluetooth;

- Web; componente che permette di gestire metodi HTTP GET, POST, PUT e DELETE.

Informazioni più dettagliate sui componenti possono essere trovate sul sito ufficiale di AI, all'indirizzo:

<http://ai2.appinventor.mit.edu/reference/components/>.

Esempio pratico: utilizzo del componente notifier

Ora che abbiamo completato anche il nostro excursus sui componenti in MIT AI, passiamo, come di consueto al nostro esempio pratico, sfruttando alcune delle conoscenze acquisite. Ci proponiamo di realizzare un'app molto semplice che, alla pressione di un tasto, visualizzi un messaggio all'utente.

Creiamo quindi un nuovo progetto, come visto nella precedente puntata e iniziamo ad collocare sul layout i componenti che ci occorrono.

Per realizzare quanto abbiamo sopra indicato ci servono i seguenti componenti:

- un componente Button;
- un componente Notifier.



Fig. 14 - Blocco When Btn_UserButton.Click.

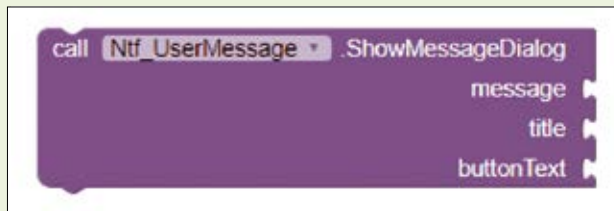


Fig. 15 - Blocco Call Ntf_UserMessage.ShowDialog.

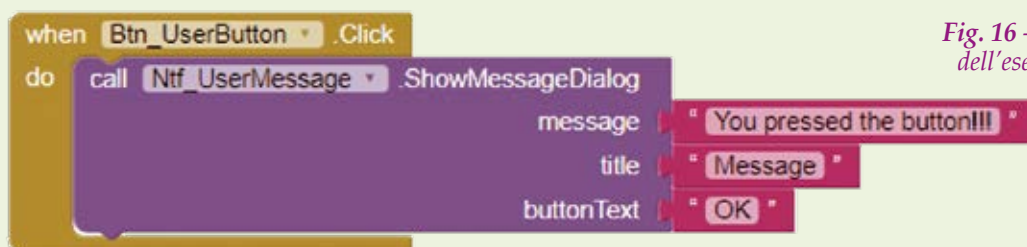


Fig. 16 - Codice grafico dell'esempio pratico.

Selezioniamoli dalla "Components Palette" (si trovano entrambi sotto il gruppo "User Interface"), posizioniamoli sul layout e poi cambiamone i nomi, rispettivamente in Btn_UserButton e Ntf_UserMessage. Inoltre cambiamo la proprietà Text del pulsante in "Press Button".

Una volta che avrete fatto ciò dovreste ottenere un risultato simile a quello di Fig. 13.

Passiamo a questo punto sulla sezione "Blocks" e scriviamo il codice grafico per l'implementazione della nostra app. Per realizzare quanto detto in precedenza, selezioniamo il blocco When Btn_UserButton.Click, presente tra i blocchi del componente button, ed il blocco Call Ntf_UserMessage.ShowDialog, tra i blocchi del componente notifier (Fig. 14 e Fig. 15) e trasciniamoli all'interno dell'area di lavoro.

A questo punto inseriamo il blocco di notifica all'interno dell'evento Click ed aggiungiamo tre costanti stringa (Text → Text String) connesse ai tre connettori del blocco di notifica, inserendo le stringhe indicate in Fig. 16.

In questo modo alla pressione del tasto, il componente notifier presenterà un pop-up message dal titolo "message" che mostra la stringa "You pressed the button!!!" e contenente un pulsante di conferma di lettura "OK".

Il risultato che si ottiene con l'emulatore è visibile nella Fig. 17.

Conclusioni

Siamo così arrivati al termine della seconda puntata, nella quale abbiamo approfondito i dettagli della programmazione grafica, illustrando strutture e dati e strutture di controllo di App Inventor e dato un primo sguardo d'insieme ai

componenti, oltre ad aver presentato il solito progetto pratico.

Dalla prossima puntata inizieremo una serie di approfondimenti sui vari componenti presenti in MIT App Inventor, corredata da diversi esempi pratici di utilizzo. ■

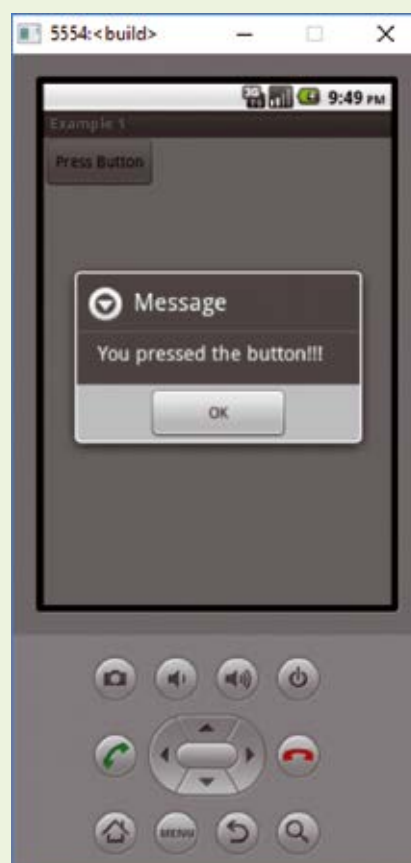


Fig. 17
Risultato
ottenuto
tramite l'uso
dell'emulatore.



MIT
APP INVENTOR

3

di FRANCESCO FICILI

Proseguiamo lo studio e la conoscenza di App Inventor, il tool di sviluppo per applicazioni Android creato dal MIT e basato su un linguaggio di programmazione grafico. In questa terza puntata analizzeremo due interessanti famiglie di componenti, ossia Storage e Sensors.

Nella scorsa puntata di questo corso ci siamo lasciati dopo aver introdotto il concetto di componente in MIT App Inventor (i componenti sono gli elementi che permettono di aggiungere funzionalità all'app in sviluppo...) ed aver constatato e sperimentato quanto potente sia questo strumento, che ci permette sostanzialmente di creare l'interfaccia grafica della nostra app, di accedere alle varie funzionalità specifiche del nostro smartphone e di utilizzare i vari sensori che si trovano al suo interno e compongono la dotazione hardware. Abbiamo anche avuto modo di conoscere le interfacce di comunicazione presenti all'interno degli smartphone e, a contorno della definizione del

concetto di componenti, con un esempio pratico siamo riusciti a sperimentare l'utilizzo del componente **notifier**, che permette di notificare all'utente un messaggio al verificarsi di un certo evento.

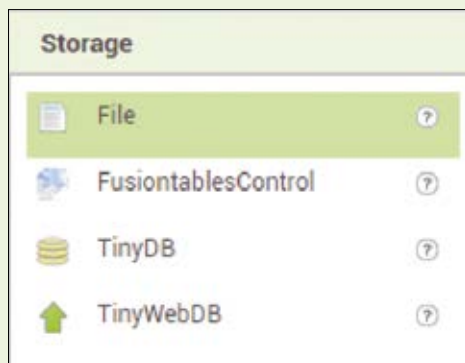


Fig. 1 - Famiglia di componenti storage.

In questa puntata riprendiamo da dove ci siamo fermati e analizziamo in dettaglio due nuove famiglie di componenti di App Inventor: la famiglia di componenti **Storage** e la famiglia di componenti **Sensors**.

Anche in questa puntata, come nelle precedenti, per brevità utilizzeremo spesso l'acronimo AI per dire App Inventor.

I componenti Storage

La famiglia di componenti Storage contiene componenti per la gestione di file e dei data-base; in essa è inoltre presente un componente per la gestione delle Google Fusion Tables, che tuttavia non tratteremo in questo corso. La **Fig. 1** riporta una schermata che raggruppa la famiglia di componenti Storage nell'IDE di App Inventor.

Ciò detto, passiamo adesso a un'analisi più di dettaglio dei singoli componenti ed in particolare

di proprietà, eventi e metodi che li caratterizzano. Ricordiamo anche brevemente la classificazione dei colori per i behaviours dei componenti:

- Proprietà: verde;
- Eventi: beige;
- Metodi: viola.

File Storage

Si tratta di un componente non visibile (non-visible component) che permette di effettuare operazio-



Fig. 2 - Layout della app di esempio per l'utilizzo del componente File.

PROPRIETÀ	
SINTASSI	DESCRIZIONE
Nessuna	-
EVENTI	
SINTASSI	DESCRIZIONE
GotText(text text)	Evento che indica che il contenuto di un dato file è stato letto e fornisce il risultato della lettura tramite il parametro 'text'.
METODI	
SINTASSI	DESCRIZIONE
AppendToFile(text text, text fileName)	Appende il parametro 'text' al file puntato dal parametro 'fileName'.
Delete(text fileName)	Cancella il file puntato dal parametro 'fileName'.
ReadFrom(text fileName)	Legge il contenuto del file puntato dal parametro 'fileName'.
SaveFile(text text, text fileName)	Salva il contenuto del parametro 'text' sul file puntato dal parametro 'fileName'.

Tabella 1 - Proprietà, Eventi e Metodi del componente File.

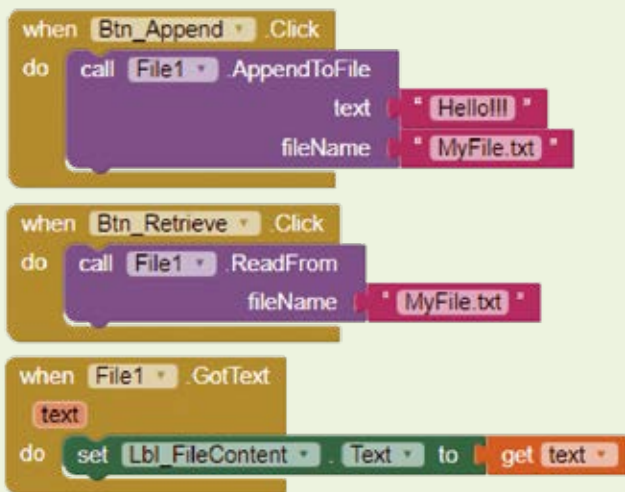


Fig. 3 - Codice grafico per l'app di esempio d'uso del componente File.

ni di salvataggio e lettura su un generico file. Per impostazione predefinita, il componente lavora sulla memoria privata dell'app in esecuzione, ma se il path di salvataggio inizia con il simbolo slash ("/"), allora il file viene creato rispetto alla directory relativa */sdcard*. Proprietà, eventi e metodi del componente sono riportati sinteticamente in **Tabella 1**. Vediamo un semplice esempio pratico che ci faccia comprendere come funziona il componente File: creiamo un nuovo progetto in AI e aggiungiamo i seguenti componenti:

- 2 componenti Button (chiamati Btn_Append e Btn_Retrieve);
- 1 componente Label (chiamato Lbl_FileContent);
- 1 componente File.

A questo punto il layout della nostra nuova app dovrebbe somigliare a quello riportato nella **Fig. 2**.

Ciò che ci proponiamo di fare è semplicemente di salvare la stringa "Hello!!!" su file alla pressione del tasto "Append" e di recuperarla e stamparla sulla label alla pressione del tasto "Retrieve".

Passiamo ora alla schermata "Blocks" ed inseriamo il codice grafico riportato in **Fig. 3**. Come potete vedere sempre dalla **Fig. 3**, abbiamo utilizzato i due metodi "AppendToFile" e "ReadFrom", collegati agli eventi "Button.Click" dei pulsanti Btn_Append e Btn_Retrieve, rispettivamente per salvare e leggere dal file "MyFile.txt".

Il metodo di lettura è asincrono, nel senso che non restituisce immediatamente il testo letto, ma bisogna utilizzare l'evento asincrono "GotText" per



Fig. 4 - Esecuzione dell'app di esempio tramite l'emulatore.

PROPRIETÀ	
SINTASSI	DESCRIZIONE
Nessuna	-
EVENTI	
SINTASSI	DESCRIZIONE
Nessuno	-
METODI	
SINTASSI	DESCRIZIONE
ClearAll()	Cancella tutti i dati contenuti nel DB.
ClearTag(text tag)	Cancella la entry associata al tag passato come parametro.
any GetTags()	Ritorna la lista di tutti i tags contenuti nel DB.
any GetValue(text tag, any valueIfTagIsNotThere)	Ritorna il valore immagazzinato in corrispondenza del tag passato come parametro. Se il tag non è presente ritorna il valore associato al parametro "ValueIfTagIsNotThere".
StoreValue(text tag, any valueToStore)	Salva permanentemente il valore passato tramite il parametro "ValueToStore" associato tag passato tramite il relativo parametro.

Tabella 2 - Proprietà, Eventi e Metodi del componente TinyDB.

accedere ai dati recuperati dal file stesso. Possiamo testare questa app usando un comune smartphone oppure l'emulatore: una volta avviata, premendo una volta sul tasto "Append" e la successiva sul tasto "Retrieve" si otterrà il risultato illustrato in Fig. 4.

Come avrete potuto notare da questo esempio applicativo, il componente File è molto utile per salvare informazioni in maniera non volatile ed è estremamente semplice da utilizzare, tuttavia può risultare poco efficiente quando con esso si debbano gestire grosse quantità di dati; in tali situazioni sono più efficienti i componenti database.

TinyDB e TinyWebDB

MIT App Inventor mette a disposizione due tipi differenti di gestione dei DB: TinyDB e TinyWebDB. Il primo è un semplice servizio di storage locale che permette di salvare dati associandoli a dei tag, in maniera da agevolare la successiva fase di recupero. Ogni app ha una sezione di memoria riservata per lo storage locale alla quale TinyDB si appoggia, tuttavia non è possibile utilizzarla per scambiare dati tra diverse app, ma è possibile sfruttarla per scambiare dati tra diversi screen in una app multi screen.

Analizziamo, come di consueto, eventi, metodi e proprietà di questo componente, aiutandoci con la Tabella 2.

Come detto in precedenza, AI implementa anche un componente che permette di gestire Database Remoti, chiamato TinyWebDB: si tratta di un set di servizi di comunicazione con un Webservice che gestisce un database, molto vantaggioso nel caso in cui ci sia la necessità di scambiare dati tra app diverse o tra dispositivi diversi. Chiaramente occorre

anche impostare la parte lato server. Non ci dilungheremo nella trattazione di questo componente, limitandoci, in questo corso, ad evidenziarne le potenzialità, analizzandone eventi, metodi e proprietà; questi sono meglio descritti nella Tabella 3. Passiamo ora ad un semplicissimo esempio di uti-

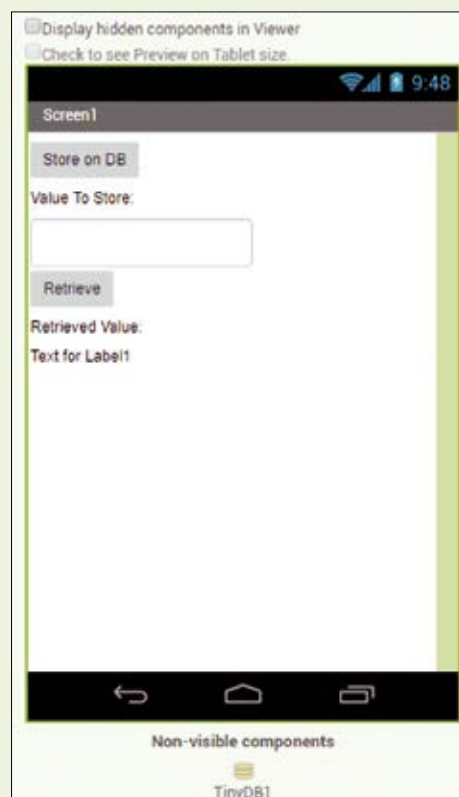


Fig. 5 - Layout dell'app relativo all'esempio di utilizzo del componente TinyDB.

PROPRIETÀ	
SINTASSI	DESCRIZIONE
ServiceURL	Questa proprietà serve ad impostare la URL del webservice con il quale il componente comunica per la gestione del DB
EVENTI	
SINTASSI	DESCRIZIONE
GotValue(text tagFromWebDB, any valueFromWebDB)	Questo evento informa che il valore richiesto al Webservice è stato recuperato e viene passato insieme al relativo tag.
ValueStored()	Questo evento informa che il Webservice ha completato una precedente richiesta di salvataggio.
WebServiceError(text message)	Questo evento indica che il Webservice ha segnalato un errore.
METODI	
SINTASSI	DESCRIZIONE
GetValue(text tag)	Questo metodo serve ad inviare una richiesta di lettura del dato marcato dal tag passato come parametro al DB Webservice.
StoreValue(text tag, any valueToStore)	Questo metodo serve ad inviare una richiesta di scrittura del dato marcato dal tag passato come parametro al DB Webservice.

Tabella 3 - Proprietà, eventi e metodi del componente TinyWebDB.

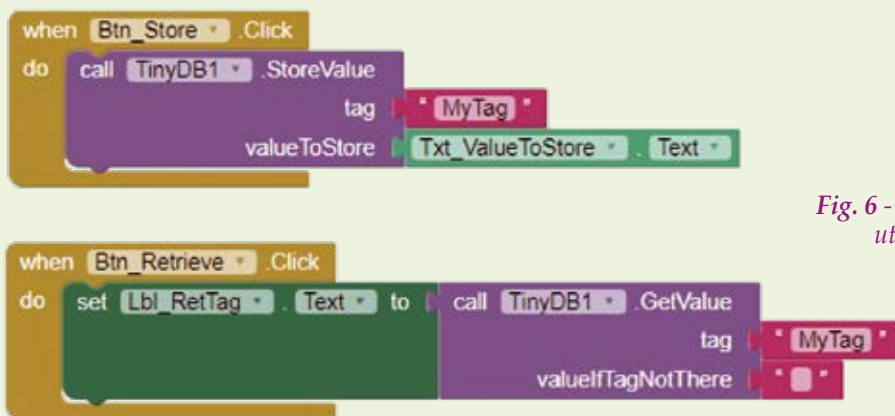


Fig. 6 - Codice grafico relativo all'esempio di utilizzo del componente TinyDB.

lizzo del componente TinyDB. Creiamo dunque una nuova app e aggiungiamo i seguenti componenti:

- 2 componenti Button (chiamati Btn_Store e Btn_Retrieve);
- 3 componenti Label (chiamati Lbl_Label1, Lbl_Label2, Lbl_ReqTag);
- 1 componente TextBox (chiamato Txt_ValueToStore);
- 1 componente TinyDB.

L'esempio è del tutto simile al precedente, ma stavolta salveremo il contenuto della textbox mar-

candolo con un tag sul data-base e lo recupereremo utilizzando il medesimo tag, per poi stamparlo sulla label.

Una volta completate le operazioni descritte in precedenza, il vostro layout dovrebbe essere del tutto simile a quello riportato nella Fig. 5.

A questo punto diamo uno sguardo al codice grafico di App Inventor, riportato nella Fig. 6. Come potete vedere, tale codice è molto semplice: alla pressione del pulsante Btn_Store viene salvato il valore del campo text del textbox, associandolo al tag "MyTag", mentre alla pressione del pulsante Btn_Retrieve viene interrogato il data-base alla ricerca del dato contrassegnato dal tag "MyTag" e il risultato viene scritto sulla label Lbl_reqTag. Il risultato ottenuto con l'emulatore è riportato nella finestra di dialogo illustrata nella Fig. 7.



Fig. 7 - Risultato ottenuto tramite l'emulatore.

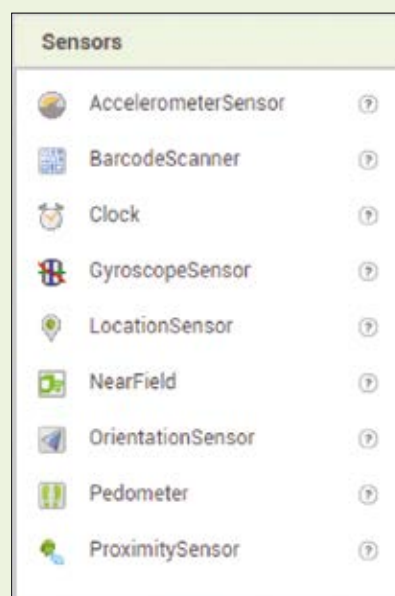


Fig. 8 - Famiglia di componenti Sensors.

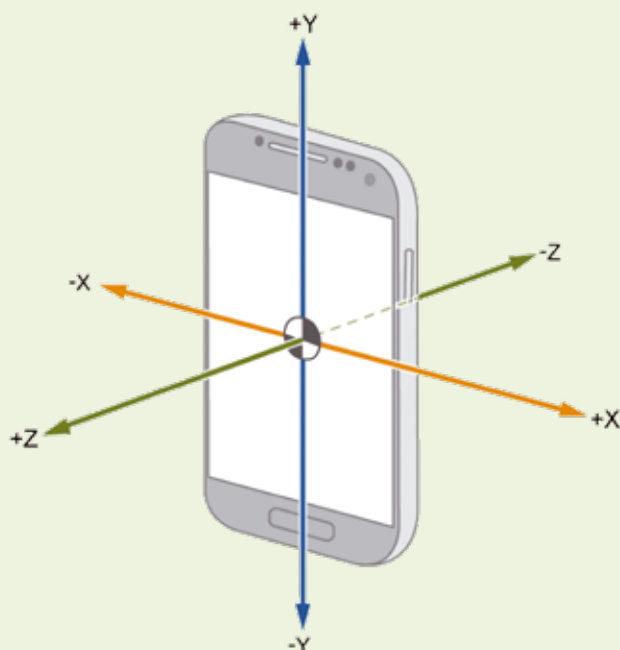


Fig. 9 - Tipica configurazione degli assi di un accelerometro, su uno smartphone Android.

I componenti Sensors

Passiamo ora all'analisi di un'altra famiglia di componenti: la famiglia di componenti Sensors. All'interno di questo gruppo si trovano componenti per la gestione dei vari sensori che tipicamente si trovano all'interno di un comune smartphone, sia fisici (come giroscopi, sensori di prossimità, GPS, ecc.), che virtuali (come ad esempio il pedometer, che in genere non rappresenta un dispositivo fisico reale, ma piuttosto una specifica elaborazione dei

dati provenienti da un altro sensore, come ad esempio un accelerometro).

In Fig. 8 è riportato uno screenshot che rappresenta i componenti presenti in questa specifica famiglia. In questa trattazione noi ci limiteremo all'analisi di alcuni componenti di questa famiglia (i componenti Accelerometer, Orientation e Pedometer), lasciando ai lettori più volenterosi l'approfondimento sugli altri.

Accelerometro

Questo componente è in grado di interfacciare l'accelerometro interno, rilevando accelerazioni in unità del SI (m/s^2) oppure condizioni di "shaking". Le accelerazioni sono rilevate sui tre assi: XAccel, YAccel, ZAccel. La Fig. 9 riporta una tipica configurazione degli assi di misura di un accelerometro in

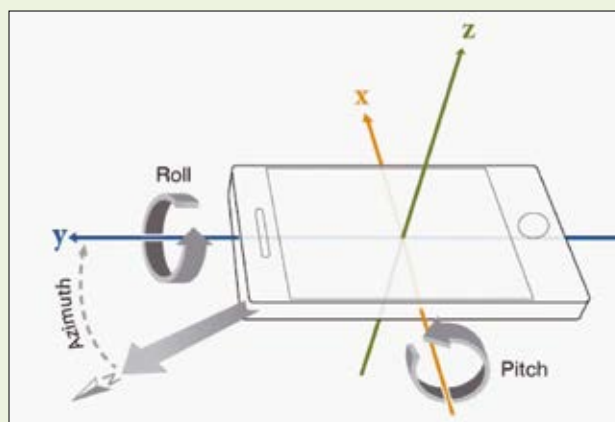


Fig. 10 - Angoli di roll, pitch e azimuth in un classico dispositivo android.

PROPRIETÀ	
SINTASSI	DESCRIZIONE
Available	Proprietà che indica se il sensore è disponibile sul dispositivo.
Enabled	Proprietà che indica se il sensore è abilitato.
MinimumInterval	Intervallo minimo di rilevamento (in ms) tra eventi di shaking.
Sensitivity	Proprietà che specifica quanto è sensibile l'accelerometro. La codifica è su tre valori: 1 = bassa, 2 = moderata, 3 = alta.
XAccel	Valore dell'accelerazione sull'asse X.
YAccel	Valore dell'accelerazione sull'asse Y.
ZAccel	Valore dell'accelerazione sull'asse Z.
EVENTI	
SINTASSI	DESCRIZIONE
AccelerationChanged(number xAccel, number yAccel, number zAccel)	Evento che indica che l'accelerazione è cambiata su uno dei tre assi e contestualmente passa i valori delle accelerazioni.
Shaking()	Evento che indica che si è verificato un evento di shaking.
METODI	
SINTASSI	DESCRIZIONE
Nessuno	-

Tabella 4 - Proprietà, eventi e metodi del componente Accelerometer.

PROPRIETÀ	
SINTASSI	DESCRIZIONE
Available	Proprietà che indica se il sensore è disponibile sul dispositivo.
Enabled	Proprietà che indica se il sensore è abilitato.
Azimuth	Ritorna l'angolo di azimut del dispositivo.
Pitch	Ritorna l'angolo di pitch del dispositivo.
Roll	Ritorna l'angolo di roll del dispositivo.
Magnitude	Ritorna un numero da 0 a 1 che indica quanto il dispositivo è inclinato.
Angle	Ritorna un angolo che indica la direzione verso la quale il dispositivo è inclinato.
EVENTI	
SINTASSI	DESCRIZIONE
OrientationChanged (number azimuth, number pitch, number roll)	Evento che indica che è cambiato l'orientamento del dispositivo, e fornisce tra i parametri i tre angoli di roll, pitch ed azimut.
METODI	
SINTASSI	DESCRIZIONE
Nessuno	-

Tabella 5 - Proprietà, eventi e metodi del componente Orientation Sensor.

un dispositivo Android.

In **Tabella 4** sono riportati, come di consueto, eventi, metodi e proprietà del componente.

Orientation Sensor

Il sensore di orientamento è un componente non visibile che permette di determinare l'orientamento spaziale del dispositivo. Questo componente fornisce i seguenti tre valori, in gradi:

- Rollio (Roll): valore dell'angolo di rollio espresso in gradi;

- Beccheggio (Pitch): valore dell'angolo di beccheggio in gradi;
- Azimut: valore dell'angolo di azimut in gradi; questo angolo vale 0° in direzione nord, 180° in direzione sud, 90° in posizione est e 270 gradi in posizione ovest; le informazioni su questo angolo possono essere utilizzate per implementare bussole digitali.

In **Fig. 10** è riportata una rappresentazione grafica che chiarisce ulteriormente come sono misurati gli

PROPRIETÀ	
SINTASSI	DESCRIZIONE
Distance	Questa proprietà ritorna la distanza approssimativa percorsa in metri.
ElapsedTime	Ritorna il tempo trascorso in millisecondi da quando il pedometro è stato attivato.
SimpleSteps	Ritorna il numero di "Simple Step" eseguiti da quando il pedometro è stato attivato.
StopDetectionTimeout	Questa proprietà serve a settare la durata in millisecondi del timeout di idle, ossia il tempo trascorso il quale senza che vengano rilevati dei passi il componente passa in "stopped" state.
StrideLength	Setta la lunghezza del passo medio in metri.
WalkSteps	Ritorna il numero di "Walk Step" eseguiti da quando il pedometro è stato attivato. Si definisce "Walk Step" il passo tipico eseguito per muoversi in avanti.
EVENTI	
SINTASSI	DESCRIZIONE
SimpleStep(number simpleSteps, number distance)	Evento che indica l'esecuzione di un "Simple Step".
WalkStep(number walkSteps, number distance)	Evento che indica l'esecuzione di un "Walk Step".
METODI	
SINTASSI	DESCRIZIONE
Pause()	Mette il pedometro in pausa.
Reset()	Resetta tutti i contatori.
Resume()	Riavvia il pedometro.
Save()	Salva tutti i contatori del pedometro in memoria non volatile.
Start()	Avvia il pedometro.
Stop()	Arresta il pedometro.

Tabella 6 - Proprietà, eventi e metodi del componente pedometer.

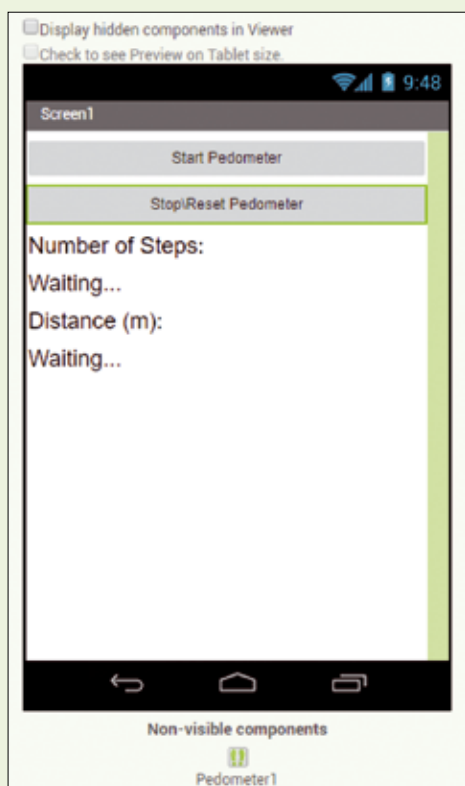
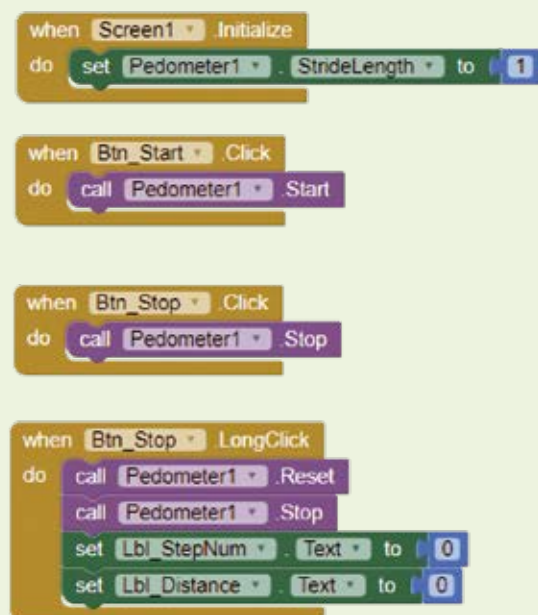


Fig. 11 - Layout dell'applicazione di esempio

angoli di roll, pitch ed azimuth. Anche in questo caso esaminiamo eventi, metodi e proprietà, aiutandoci con la **Tabella 5**.



Pedometer

L'ultimo componente della famiglia sensor che analizziamo in questa puntata del corso è il Pedometer (o pedometro): si tratta di un sensore virtuale, che sfrutta le informazioni provenienti dall'accelerometro interno per determinare se la persona che porta con sé lo smartphone ha compiuto dei passi; ciò viene ottenuto analizzando l'accelerazione in una combinazione di direzioni che fa presupporre che si stia camminando, perché corrisponde al movimento in avanti e dal basso in alto e viceversa, il tutto contemporaneamente.

Inoltre, sempre con il componente pedometer, configurando opportunamente un parametro (strideLength) è possibile anche stimare la distanza percorsa.

Esaminiamo anche in questo caso eventi, metodi e proprietà, aiutandoci con la **Tabella 6**, che li raccoglie e li descrive.

Esempio pratico sull'uso dei componenti Sensors

Passiamo a questo punto al nostro consueto esempio pratico, utile a sperimentare i concetti che abbiamo esposto. Per l'esempio abbiamo deciso di utilizzare l'ultimo componente descritto (ossia il Pedometer) per realizzare una semplice applicazione contapassi, con in più una misura approssimativa della distanza percorsa in metri. Per realizzare questa applicazione posizioniamo i seguenti elementi grafici:

- 2 Buttons (Btn_Start e Btn_Stop);

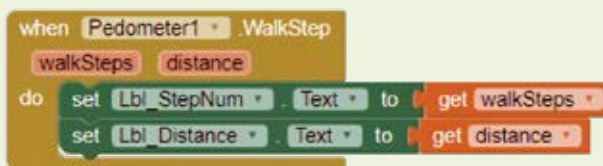


Fig. 12 - Codice grafico dell'applicazione di esempio

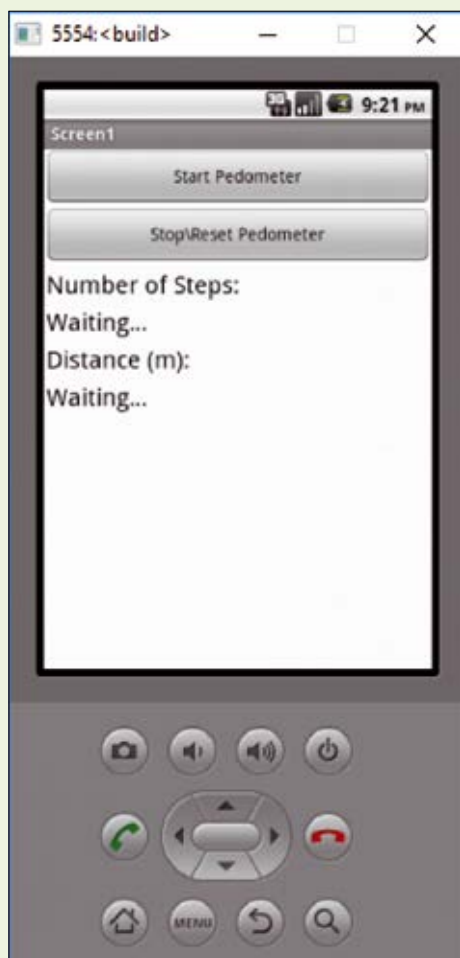


Fig. 13 - Screenshot dell'applicazione di esempio in emulazione.

- 4 Labels (Lbl_Label1, Lbl_Label2, Lbl_StepNum, Lbl_Distance);
- 1 componente pedometer.

A questo punto organizziamo il layout come proposto in Fig. 11 e passiamo sulla Block view.

La realizzazione dell'app, sfruttando le potenzialità del componente pedometer, risulta piuttosto semplice; infatti sfruttiamo gli eventi button.click sui pulsanti Btn_Start e Btn_Stop rispettivamente per avviare e arrestare il contapassi (invocando i metodi **Start()** e **Stop()**) ed inoltre utilizziamo l'evento Button.LongClick per resettare i contatori (chiamando in sequenza i metodi **Reset()** e **Stop()** e resettando le label così da riportare il valore 0).

Poi sfruttiamo l'evento Pedometer.WalkStep per contare i passi, settando ad ogni occorrenza dell'evento stesso i valori delle due label Lbl_StepNum ed

Lbl_Distance, pari ai valori dei parametri WalkSteps e Distance che possiamo recuperare tramite gli appositi metodi di get.

Il codice grafico che implementa quanto spiegato in questo esempio pratico è riportato nella Fig. 12. Invece nella Fig. 13 trovate uno screenshot dell'applicazione che gira in emulazione sull'emulatore, mentre nella Fig. 14 riportiamo un ulteriore screenshot ma stavolta catturato da un dispositivo reale, durante il funzionamento con l'app in esecuzione.

Conclusioni

In questa terza puntata abbiamo iniziato ad analizzare più in dettaglio una delle caratteristiche più interessanti di MIT Appinventor, ossia i componenti, concentrandoci in particolare sulle famiglie Storage e Sensors.

Nelle prossime puntate continueremo su questa linea, studiando nuovi componenti (l'elenco completo dei componenti di App Inventor lo trovate nella seconda puntata del corso) passando anche in rassegna componenti che permettono di interagire con dell'hardware esterno allo smartphone.

Quindi vi diamo appuntamento al mese prossimo! ■

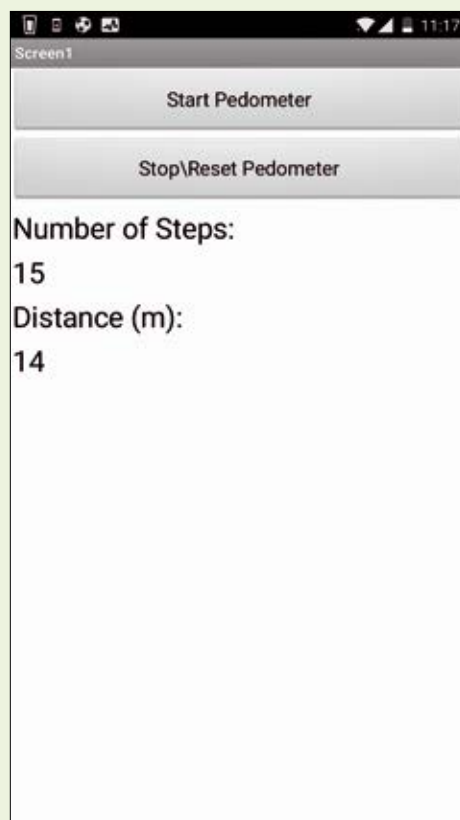


Fig. 14 - Screenshot dell'applicazione di esempio su un dispositivo reale.



MIT APP INVENTOR

4

di FRANCESCO FICILI

Proseguiamo la trattazione di MIT App Inventor, descrivendo e sperimentando due nuove e interessanti famiglie di componenti dell'ambiente di sviluppo: Media e Social.

Nella puntata precedente ci siamo lasciati dopo avere iniziato a descrivere in dettaglio alcune famiglie di componenti di App Inventor ed in particolare ci siamo concentrati sulla descrizione delle famiglie **Storage** (la famiglia contiene componenti per la gestione di file e dei data-base) e **Sensor** (all'interno di questo gruppo si trovano componenti per la gestione dei sensori tipicamente integrati in uno smartphone, sia fisici come accelerometri, giroscopi, sensori di prossimità e GPS, sia virtuali, come ad esempio il pedometro).

In questa puntata continuiamo l'analisi dei componenti di MIT App Inventor, passando in rassegna altre due famiglie di componenti: le famiglie di

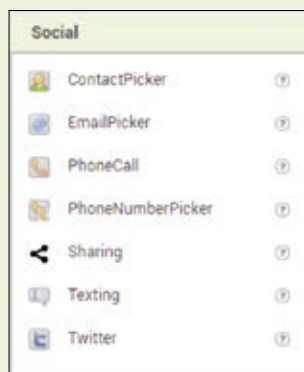


Fig. 1
Famiglia di componenti Social.

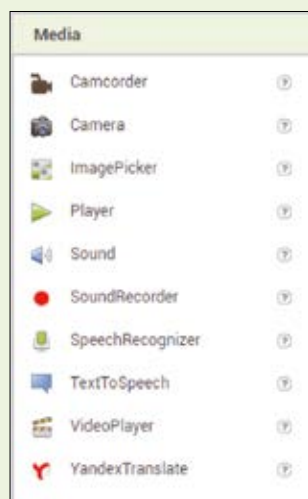


Fig. 2
Famiglia di componenti Media.

Tabella 1 - Descrizione dei componenti della famiglia Social.

COMPONENTE	DESCRIZIONE
ContactPicker	Picker che permette di scegliere un contatto tra la lista di contatti memorizzata sullo smartphone. Una volta selezionato il contatto possono venire fornite le seguenti informazioni relative: - ContactName, - EmailAddress, - EmailAddressList, - ContactUri, - PhoneNumber, - PhoneNumberList, - Picture
EmailPicker	Picker che si presenta come una textbox. Se l'utente inserisce il nome parziale di un contatto o di una email, il picker presenterà un dropdown menù di selezioni che completano la entry parziale.
PhoneCall	Componente non visibile che permette di effettuare telefonate.
PhoneNumberPicker	Componente che funziona in maniera simile al ContactPicker, ma restituisce un set inferiore di informazioni correlate: - ContactName, - PhoneNumber, - EmailAddress, - Picture
Sharing	Componente non visibile che permette di condividere files e/o messaggi tra l'app in esecuzione ed altre app installate sul dispositivo. Il componente visualizzerà un elenco di app che possono gestire le informazioni che si intende condividere e permetterà all'utente di selezionarne una (ad es. una app di email, un app di un social network, una app di messaggistica istantanea, etc..).
Texting	Componente non visibile che permette di inviare e ricevere messaggi SMS.
Twitter	Componente che permette di interfacciarsi con twitter. Il componente permette di eseguire le seguenti operazioni: - Searching Twitter for tweets or labels (SearchTwitter) - Sending a Tweet (Tweet) - Sending a Tweet with an Image (TweetWithImage) - Directing a message to a specific user (DirectMessage) - Receiving the most recent messages directed to the logged-in user (RequestDirectMessages) - Following a specific user (Follow) - Ceasing to follow a specific user (StopFollowing) - Getting a list of users following the logged-in user (RequestFollowers) - Getting the most recent messages of users followed by the logged-in user (RequestFriendTimeline) - Getting the most recent mentions of the logged-in user (RequestMentions)

componenti **Media** e **Social**.

Prima di iniziare la trattazione di queste famiglie ricordiamo che in App Inventor si chiamano "componenti" quegli elementi che permettono di aggiungere funzionalità all'app che stiamo sviluppando; i componenti sono dunque lo strumento che ci permette sostanzialmente di creare l'interfaccia grafica della nostra app, di accedere alle varie funzionalità specifiche del nostro dispositivo mobile, sia esso uno smartphone o un tablet, nonché di utilizzare i vari sensori che si trovano al suo interno e che compongono la dotazione hardware.

I componenti Social

Iniziamo subito precisando che in MIT App Inventor vengono classificati come Social tutti quei componenti che permettono di interagire con i meccanismi

social integrati nello smartphone. Questi possono essere meccanismi classici, come ad esempio le chiamate telefoniche e gli SMS, oppure più recenti e strettamente legati alla Rete, come i social network (Twitter, Whatsapp e via di seguito).

Non ci soffermeremo molto sulla descrizione di questi componenti in quanto non li utilizzeremo nell'esempio pratico presentato in questa puntata; ci limiteremo, piuttosto, a fornire una tabella riassuntiva che ne illustra le caratteristiche generali.

I componenti Media

In MIT App Inventor sono identificati come componenti Media tutti quei componenti che sono in grado di gestire applicazioni multimediali, ossia quelle che consentono di effettuare la registrazione e la riproduzione di audio e video.

Tabella 2 - Descrizione dei componenti della famiglia Media.

COMPONENTE	DESCRIZIONE
Camcorder	Componente che permette di registrare un video utilizzando il camcorder del dispositivo.
Camera	Componente che permette di scattare una foto utilizzando la camera del dispositivo.
ImagePicker	Picker che permette di selezionare un'immagine della galleria di immagini del dispositivo.
Player	Componente multimediale che permette di riprodurre una traccia audio e controllare la vibrazione del dispositivo.
Sound	Componente multimediale che permette di riprodurre un suono e controllare la vibrazione del dispositivo.
SoundRecorder	Componente multimediale che permette di registrare audio usando il mic del dispositivo.
SpeechRecognizer	Componente che permette di convertire un suono acquisito tramite il microfono in un testo.
TextToSpeech	Componente che, dato un testo in ingresso, permette di sintetizzarlo e riprodurlo tramite lo speaker del dispositivo.
VideoPlayer	Componente multimediale che permette di riprodurre video sul display del dispositivo.
YandexTranslate	Componente che permette di effettuare traduzioni tra linguaggi differenti sfruttando il servizio online Yandex.Translate. Necessita di una connessione internet per funzionare.

Tabella 3 - Eventi, metodi e proprietà del componente SpeechRecognizer

PROPRIETÀ	
SINTASSI	DESCRIZIONE
Result	L'ultimo testo generato dalla conversione.
EVENTI	
SINTASSI	DESCRIZIONE
AfterGetting(Text result)	Evento triggerato una volta che il convertitore ha effettuato la conversione in testo. L'argomento result contiene il testo risultato della conversione.
BeforeGettingText()	Evento triggerato appena prima che il convertitore venga invocato.
METODI	
SINTASSI	DESCRIZIONE
GetText()	Questo metodo chiede all'utente di parlare e converte il parlato in testo. L'evento AfterGetting(Text) viene invocato subito dopo che la conversione viene ultimata.

Tali componenti interagiscono con le interfacce audio video del dispositivo mobile, quindi con il microfono e lo speaker (altoparlante) nel caso dell'audio e con la fotocamera nel caso del video. Inoltre alcuni componenti di questa famiglia sono in grado di interagire con la galleria di immagini memorizzate nella memoria di massa del nostro

smartphone o tablet.

In Fig. 2 è riportata un'immagine raffigurante la famiglia di componenti Media nell'IDE di App Inventor. In Tabella 2 è fornita una descrizione sommaria dei vari componenti della famiglia, mentre nei paragrafi successivi forniremo una descrizione più dettagliata, corredata della solita descrizione di

Tabella 4 - Eventi, metodi e proprietà del componente TextToSpeech.

PROPRIETÀ	
SINTASSI	DESCRIZIONE
AvailableCountries	Elenca tutti i country code disponibili sul dispositivo per essere utilizzati con il componente.
AvailableLanguages	Elenca tutti i language code disponibili sul dispositivo per essere utilizzati con il componente.
Country	Country code da utilizzare con il componente TextToSpeech. Questa proprietà incide sulla pronuncia, modificando l'accento. Ad esempio un accento con country code US (Stati Uniti) sarà diverso da un accento con country code UK (Gran Bretagna).
Language	Imposta il linguaggio da utilizzare quando viene sintetizzata la voce.
Pitch	Imposta il Pitch. Il range di valori è compreso tra 0 e 2.
SpeechRate	Imposta lo SpeechRate, ossia la velocità con cui viene pronunciata la parola o la frase. Il range di valori è compreso tra 0 e 2.
EVENTI	
SINTASSI	DESCRIZIONE
AfterSpeaking(boolean result)	Evento triggerato subito dopo la riproduzione del testo.
BeforeSpeaking()	Evento triggerato subito dopo l'invocazione del metodo Speak e subito prima della riproduzione del testo.
METODI	
SINTASSI	DESCRIZIONE
Speak(text message)	Metodo che avvia la sintesi del testo passato come parametro.

Tabella 5 - Eventi, metodi e proprietà del componente YandexTranslate.

PROPRIETÀ	
SINTASSI	
Nessuna	
EVENTI	
SINTASSI	DESCRIZIONE
GotTranslation(text responseCode, text translation)	Evento triggerato quando il servizio Yandex.Translate restituisce il testo tradotto. Insieme alla traduzione viene fornito un codice di risposta, se il codice è diverso da 200 qualcosa non ha funzionato correttamente e la traduzione non è stata effettuata.
METODI	
SINTASSI	DESCRIZIONE
RequestTranslation(text languageToTranslateTo, text textToTranslate)	Metodo che permette di effettuare una richiesta di traduzione al servizio online, fornendo il target language per la traduzione e il testo da tradurre. Il linguaggio target viene fornito con language code (ad es. it = italiano, en = inglese, es = spagnolo ...). Una volta completata la traduzione, il componente triggera un evento che fornisce il risultato. Yandex cerca di determinare il linguaggio di partenza direttamente dal testo, ma è anche possibile dichiararlo esplicitamente, aggiungendolo prima del linguaggio target e separandolo con uno spazio (ad es. it-en richiederà una traduzione da italiano ad inglese).

eventi, metodi e proprietà dei componenti SpeechRecognizer, TextToSpeech e YandexTranslate.

Speech Recognizer

Questo componente sfrutta l'input proveniente dal microfono integrato nello smartphone per acquisire il segnale audio e poi sfrutta le feature di speech recognizing integrate in Android per convertire il parlato in testo. In **Tabella 3** sono riportati eventi, metodi e proprietà.

Text To Speech

Il componente TextToSpeech permette di sintetizzare vocalmente un testo. È possibile anche impostare alcune proprietà, come ad esempio language e country per cambiare l'accento con cui viene sintetizzata la voce. In **Tabella 4** sono riportati, come di consueto eventi, metodi e proprietà.

Yandex translate

Questo componente permette di tradurre parole o frasi tra lingue differenti, utilizzando il servizio online di traduzione Yandex Translate. Il compo-

nente lavora in maniera del tutto asincrona, inviando una richiesta al servizio online tramite un metodo e fornendo il risultato tramite un apposito evento. Informazioni aggiuntive, come la lista dei linguaggi supportati, significato dei language code e altro, possono essere recuperate all'indirizzo web:

<http://api.yandex.com/translate/>

In **Tabella 5** sono riportati eventi, metodi e proprietà del componente.

Esempio pratico: traduttore vocale simultaneo

Sfruttando i tre componenti appena descritti nel dettaglio creiamo un semplice esempio pratico che implementa un traduttore vocale simultaneo, ossia una app in grado di acquisire un testo (parola singola o frase) da tradurre, effettuare la traduzione e riprodurla. In questa semplice implementazione faremo sì che il linguaggio di partenza sia sempre l'italiano, mentre il target language sarà selezionabile tra una serie di linguaggi a scelta da un elenco

Fig. 3 - Sistemazione degli elementi del layout dell'app di esempio.

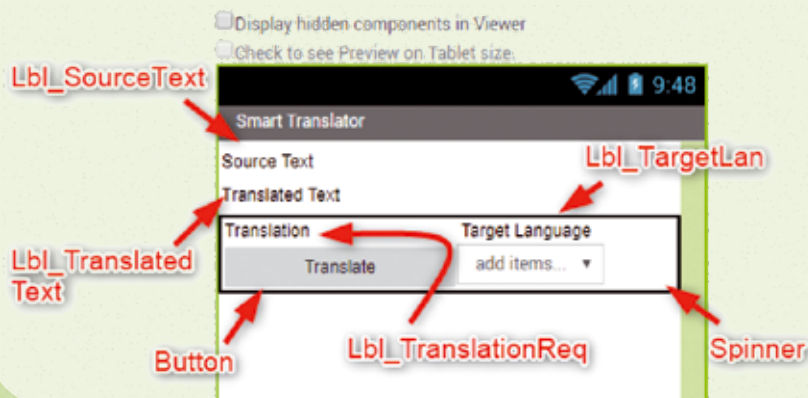
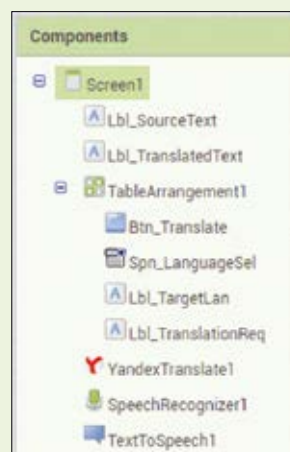


Fig. 4 - Alberatura dei componenti del layout.



apposito, ma l'app può essere facilmente estesa ad una casistica più generale.

Partiamo, come sempre, dal layout, punto di partenza dello sviluppo di qualsiasi app con AI. Ci servono i seguenti componenti:

- 4 Labels, chiamate rispettivamente Lbl_SourceText, Lbl_TranslatedText, Lbl_TranslationReq, Lbl_TargetLan
- Un pulsante Btn_Translate
- Uno spinner Spn_LanguageSel
- Un TableArrangement
- Un componente TextToSpeech
- Un componente SpeechRecognition
- Un componente YandexTranslate

A questo punto sistemiamo i vari componenti nel nostro layout, come riportato nella Fig. 3.

Nella Fig. 4 trovate riportata l'alberatura (la struttura ad albero, per intenderci...) dei componenti del nostro progetto App Inventor.

A questo punto dobbiamo impostare alcune proprietà dello **spinner**: lo spinner è un elemento che permette di far comparire una finestra di pop-up con una lista di elementi tra cui scegliere. Tali elementi possono essere impostati tramite la proprietà **ElementFromString**, settabile sia dal block che dal designer editor.

Noi, nel nostro esempio pratico utilizzeremo lo spinner per creare l'elenco dei "target language", ossia delle lingue nelle quali vogliamo che venga tradotto l'input vocale fornito in italiano, ossia il parlato che il microfono dello smartphone o tablet andrà a captare.

Nell'implementare il nostro traduttore desideriamo poter fornire la scelta del target language tra le seguenti lingue:

- En – Inglese,
- Fr – Francese,
- Ru – Russo,



Fig. 5
Modifica della proprietà ElementFromString dello spinner.

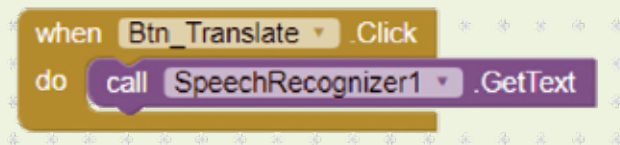


Fig. 6 - *Call del metodo GetText al verificarsi dell'evento Click.*

- Es – Spagnolo,
- De – Tedesco

Quindi modifichiamo la proprietà **ElementFromString** dello spinner come illustrato in Fig. 5, vale a dire inserendo la concatenazione di stringhe: en, de, ru, es, fr (le lingue che il traduttore dovrà gestire).

Inoltre inseriamo anche la stringa che vogliamo venga introdotta in testa al pop-up: lo facciamo inserendola nella textbox della proprietà **Prompt** ("Language Selection").

A questo punto possiamo passare al block editor



Fig. 7 - *Codice grafico che implementa la richiesta di traduzione al verificarsi dell'evento AfterGettingText.*

ed iniziare a lavorare sui behaviours; in particolare, desideriamo che alla pressione del pulsante **Btn_Translate** venga triggerata l'acquisizione del testo tramite il componente **SpeechRecognizer**, subito dopo venga effettuata la richiesta di traduzione al servizio di **Yandex** ed, una volta intercettato, l'evento **GotTranslation**, sintetizzata la traduzione. Contestualmente stamperemo sulle label **Lbl_SourceText** ed **Lbl_TranslatedText** sia il testo acquisito, sia la sua traduzione.

Per fare quanto descritto sopra, per prima cosa intercettiamo l'evento **Click** del pulsante **Btn_Translate** e chiamiamo il metodo **GetText** del componente **SpeechRecognizer** per attivare la conversione del

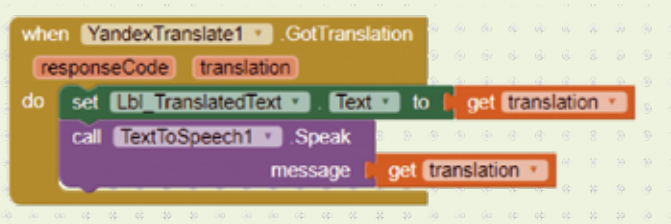


Fig. 8 - *Implementazione della parte finale del codice grafico.*

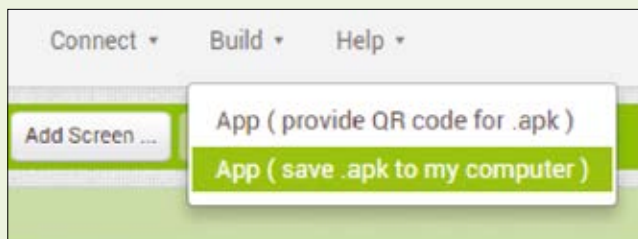


Fig. 9 - Creazione del file apk.

parlato in testo, come illustrato in Fig. 6.

A questo punto intercettiamo l'evento AfterGettingText del componente SpeechRecognizer per eseguire le seguenti operazioni:

- Impostare la proprietà language del componente TextToSpeech pari al valore selection dello spinner (questo permette di utilizzare il corretto linguaggio nella riproduzione),
- Invocare il metodo RequestTranslation del componente YandexTranslate passando il testo da tradurre (argomento result dell'evento AfterGettingText) ed i language code da utilizzare (in questo caso facciamo una join tra la stringa fissa "it-" e la selection corrente dello spinner).
- Settare la proprietà Text della label Lbl_SourceText pari al testo da tradurre (sempre argomento result dell'evento AfterGettingText), in modo da stampare sulla label il testo da tradurre, appena acquisito tramite il componente SpeechRecognizer.

Il codice grafico che implementa quanto descritto sinora è riportato nella Fig. 7.

A questo punto non rimane altro da fare che intercettare l'evento GotTranslation del componente YandexTranslate ed eseguire le seguenti due operazioni:

- settare la label Lbl_TranslatedText pari all'argomento translation dell'evento GotTranslation, in modo da visualizzare il testo tradotto sulla

rispettiva label;

- invocare il metodo Speak del componente TextToSpeech passando anche in questo caso l'argomento translation dell'evento GotTranslation in modo da riprodurlo vocalmente.

Quando descritto viene implementato tramite lo snippet grafico riportato in Fig. 8.

Per testare questo specifico esempio pratico non potete utilizzare l'emulatore; vi suggeriamo quindi di eseguire le vostre prove con uno smartphone reale, utilizzando uno dei metodi di emulazione tramite aiCompanion descritti nella prima puntata (il metodo che impiega il canale WiFi è di gran lunga il più efficiente e comodo da utilizzare).

In alternativa potete anche generare il file apk e scaricarlo sul vostro Smartphone, per poi installarlo ed eseguirlo su di esso. Per fare ciò, dovete selezionare l'opzione "APP (Save .apk on my computer)" dalla voce Build che trovate nella top bar, come illustrato in Fig. 9.

Nella Fig. 10 è riportata la fase di generazione dell'apk che consegue a tale operazione.

Conclusioni

In questa quarta puntata del nostro corso abbiamo passato in rassegna altri due interessanti componenti di MIT App Inventor, ossia i componenti Media e Social; per farvi impraticare con essi, come di consuetudine vi abbiamo presentato un esempio pratico dove vengono utilizzati tre componenti Media: stavolta si è trattato di implementare un semplice traduttore vocale simultaneo, ossia una app in grado di acquisire un testo (parola singola o frase) da tradurre, effettuare la traduzione e riprodurre il parlato corrispondente tramite l'altoparlante del device mobile.

Nella prossima puntata continueremo la nostra analisi occupandoci della descrizione di nuovi componenti di App Inventor.

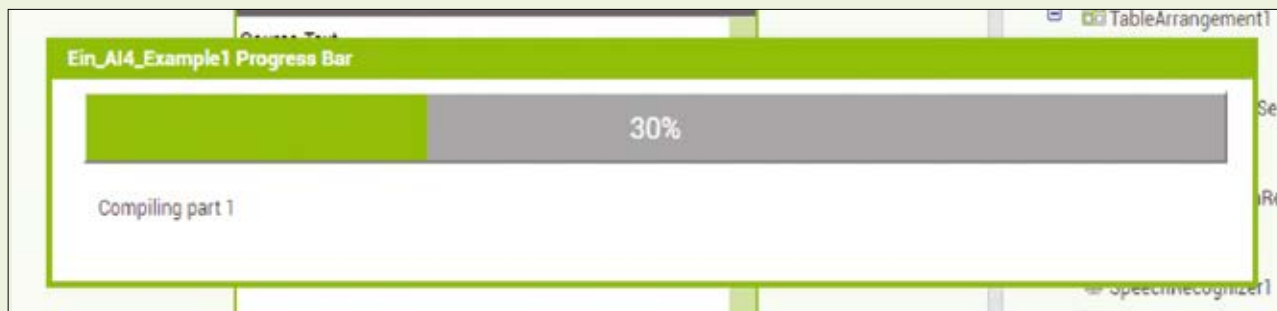


Fig. 10 - Fase di generazione del file apk.



MIT APP INVENTOR

5

di FRANCESCO FICILI

Continuiamo il nostro viaggio alla scoperta del tool di sviluppo per applicazioni Android basato su un linguaggio di programmazione completamente grafico. In questa puntata analizzeremo la gestione delle liste e delle procedure, nonché delle animazioni, tramite la famiglia di componenti drawing and animation.

Nella precedente puntata di questo nostro corso abbiamo portato avanti l'analisi dei componenti di App Inventor (ricordiamo che i componenti sono gli elementi che permettono di aggiungere funzionalità all'app in fase di sviluppo con App Inventor) descrivendo le famiglie di componenti **media** e **social** e proponendo i consueti esempi applicativi propedeutici al consolidamento delle nozioni teoriche acquisite. In questa puntata continueremo la nostra analisi concentrandoci sulla gestione delle stringhe e delle liste, descrivendo come possono essere create le

procedure e, infine, analizzando una nuova famiglia di componenti: **drawing** e **animations**.

Gestire le Stringhe in AI

App Inventor (AI, come lo spesso lo abbreviamo in questo corso) mette a disposizione una serie di behaviours specializzati per la gestione delle stringhe, che prendono il nome di "Text Block", che nella block palette sono localizzati tra i blocchi **built-in**. I blocchi principali di questo gruppo sono riportati nella **Fig. 1**, mentre nella **Tabella 1** riportiamo i più significativi, ciascuno correlato con l'operazione corrispondente e con la relativa descrizione. App Inventor permette di compiere una serie

piuttosto ampia e diversificata di operazioni sulle stringhe, tra cui:

- creazione di costanti di tipo stringa;
- unione di due o più stringhe;
- calcolo della lunghezza di una stringa;
- verifica che una stringa sia o meno vuota;
- confronto tra due stringhe;
- eliminazione di spazi (trim);
- conversione Maiuscolo/Minuscolo;

ed altro ancora. Come abbiamo visto in precedenza, App Inventor identifica i blocchi di gestione stringhe con il colore violetto.

Vediamo di seguito quali sono le operazioni più



*Fig. 1
Principali
blocchi del
gruppo Text.*

SIMBOLO	OPERAZIONE	DESCRIZIONE
	Costante stringa	Questo blocco crea una costante di tipo stringa. È possibile digitare il contenuto della stringa direttamente dentro il box tra le virgolette.
	Unione stringhe	Tramite questo blocco è possibile unire due o più stringhe.
	Lunghezza stringhe	Questo blocco restituisce la lunghezza di una stringa in numero di caratteri. Un altro blocco simile denominato "is empty" verifica se una determinata stringa è vuota.
	Confronto tra stringhe	Questo blocco permette di eseguire il confronto tra due stringhe, determinando se sono uguali. È possibile inoltre verificare se una data stringa è maggiore o minore di un'altra (dal punto di vista logico-alfabetico).
 	Inizia o contiene	Questi due blocchi consentono di verificare se una stringa inizia per o contiene una data sottostringa. Il blocco "starts" restituisce 0 se la sottostringa non è presente, oppure la posizione del primo carattere della sottostringa nella stringa. Il blocco "contains" invece restituisce semplicemente true o false a seconda che la sottostringa sia presente o meno.
 	Dividere stringhe	Tramite il blocco split è possibile effettuare la divisione di una stringa in funzione di un determinato separatore (at). Il blocco split at spaces divide la stringa in funzione del separatore spazio. In entrambi i casi viene restituita una lista di elementi.
	Tagliare stringhe	Tramite questo blocco è possibile ricavare una sottostringa da una stringa partendo da un determinato offset (start) e per una determinata lunghezza (length).
	Sostituire in una stringa	Tramite questo blocco è possibile sostituire un elemento (segment) in una data stringa con una stringa sostitutiva (replacement). La sostituzione viene fatta con tutte le occorrenze di "segment".

Tabella 1 - Operazioni più comuni sulle stringhe con AI.

comuni che è possibile effettuare sulle stringhe di App Inventor, aiutandoci con la **Tabella 1**, dove abbiamo spiegato quelle più rilevanti.

Gestire le Liste in AI

Uno degli elementi più utilizzati nei vari linguaggi di programmazione moderni sono le liste; in realtà la maggior parte dei linguaggi utilizza degli elementi più semplici che sono gli array e le matrici (si tratta di vettori mono e bi-dimensionali), e le liste vengono generate tramite strutture più complesse. In App Inventor le liste sono definite come un insieme di elementi contraddistinti da un indice numerico che identifica univocamente il dato all'interno

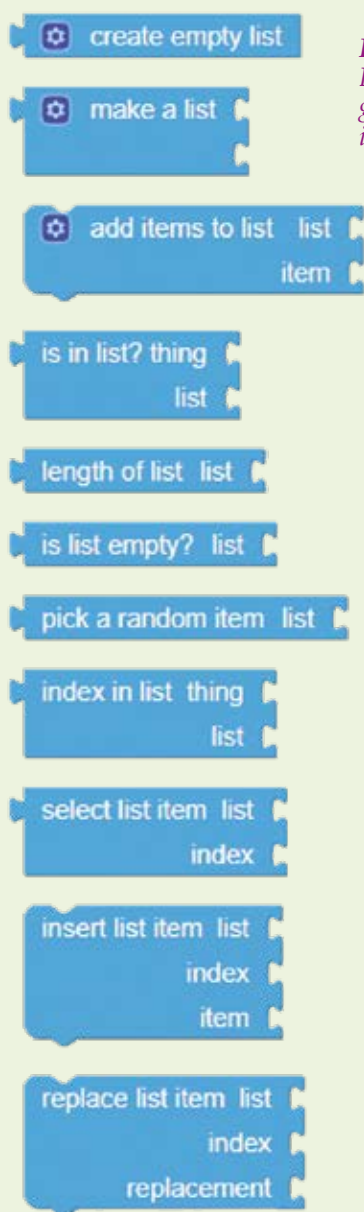
della lista; il primo elemento di una lista ha indice 1, a differenza della stragrande maggioranza dei linguaggi di programmazione, che identifica questo elemento con l'indice 0.

Le più importanti operazioni che è possibile eseguire sulle liste in App Inventor sono:

- creazione di una lista;
- manipolazione degli elementi di una lista;
- selezione degli elementi di una lista;
- verifica dello stato di una lista (vuota, numero di elementi contenuti);
- verifica che un dato elemento sia contenuto in una lista;
- determinazione dell'indice di un elemento in una lista.

ed altro ancora. Nella **Fig. 2** sono riportati alcuni dei blocchi di gestione liste di App Inventor; come abbiamo spiegato in precedenza e come si vede in figura, App Inventor identifica i blocchi di gestione liste con il colore celeste.

Analizziamo adesso i più importanti blocchi di App Inventor per la gestione delle liste, aiutandoci con **Tabella 2**.



*Fig. 2
Blocchi
gestione liste
in AI.*

SIMBOLO	OPERAZIONE	DESCRIZIONE
	Creazione Liste	Questi blocchi consentono di creare una lista vuota o una lista con una serie di elementi.
	Aggiunta di elementi in coda ad una lista	Questo blocco consente di aggiungere nuovi elementi alla lista (è possibile aggiungere elementi cliccando sull'icona a forma di ingranaggio). I nuovi elementi saranno aggiunti in coda alla lista.
	Inserimento e rimozione di elementi da una lista	Tramite questi blocchi è possibile inserire e rimuovere elementi da una lista tramite il loro indice.
	Sostituzione di un elemento di una lista	Tramite questo blocco è possibile sostituire un elemento in una lista utilizzando l'indice per individuarlo.
	Individuazione di un elemento in lista	Questi blocchi consentono di verificare se un dato elemento è presente in una lista oppure se una lista è vuota.
	Indice di un elemento in lista	Questo blocco consente di ottenere l'indice di un dato elemento in una lista.

Tabella 2 - Principali blocchi di gestione liste in AI.

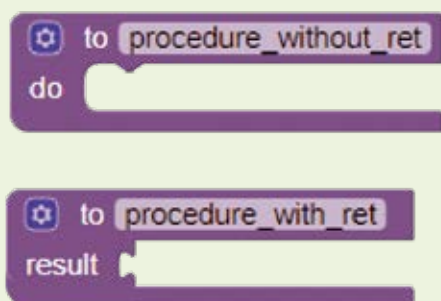


Fig. 3 - Tipi di procedure supportate in AI.

Esistono vari altri blocchi che consentono di eseguire ulteriori operazioni sulle liste, ma in questa sede non ce ne occuperemo; lasciamo a voi l'approfondimento di questo argomento.

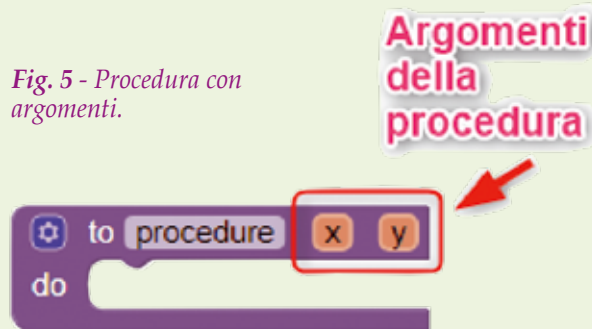
Le procedure in App Inventor

Come la maggior parte dei linguaggi di programmazione, anche in AI è possibile creare delle sub-routine, ossia delle funzioni che prendono in ingresso un set di parametri, eseguono una determinata elaborazione ed infine possono, eventualmente, ritornare un determinato valore come risultato dell'elaborazione stessa. Una volta definita, la funzione (o sub-routine) ha la caratteristica di poter essere richiamata in qualsiasi altra sezione della nostra applicazione, permettendo di realizzare alcuni principi base della programmazione, come la modularità e il riutilizzo del codice.

In App Inventor queste sub-routine prendono il nome di procedure e sono rappresentate con blocchi di colore violetto.

AI definisce due tipi principali di procedure, con va-

Fig. 5 - Procedura con argomenti.



lore di ritorno e senza valore di ritorno, come illustrato in Fig. 3. In entrambi i casi è sempre possibile, se necessario, passare degli argomenti alla procedura; questo può essere fatto semplicemente premendo sul pulsante a forma di ingranaggio presente su uno dei blocchi procedure visto in precedenza ed effettuando il drag&drop dei blocchi input; l'operazione è schematizzata nella Fig. 4. Invece nella Fig. 5 è riportata una procedura che ha due argomenti: x e y.

Le procedure devono avere nomi univoci all'interno di ogni Screen e una volta definite comparirà tra i blocchi "Procedures" l'apposito blocco per la relativa invocazione (Call Procedure), come illustrato in Fig. 6; tale blocco può essere utilizzato per invocare una procedura da un punto qualsiasi della nostra app.

Esempio pratico di procedure

Vediamo adesso come creare una semplice procedura che esegua la moltiplicazione tra due valori, passati come argomenti alla procedura stessa. Per

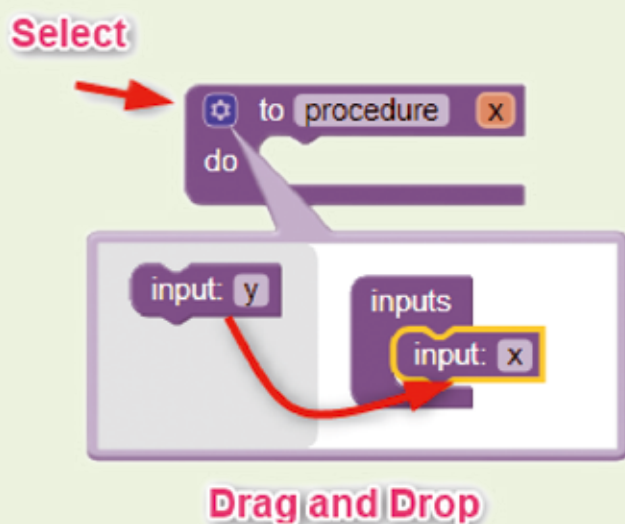


Fig. 4 - Aggiunta degli argomenti ad una procedura.

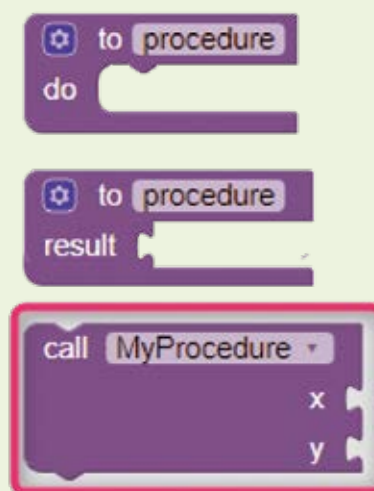


Fig. 6 - Blocco per l'invocazione di una procedura.

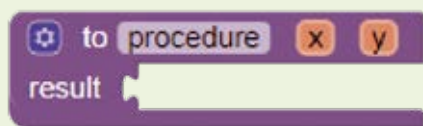


Fig. 7 - Procedura con ritorno ed argomenti x e y.

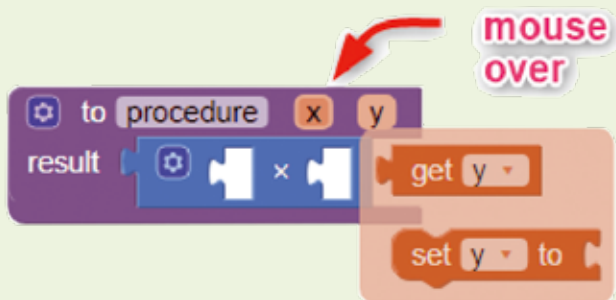


Fig. 8 - Inserimento del blocco moltiplicazione e dei blocchi get degli argomenti della procedura.

prima cosa trasciniamo all'interno del workspace un blocco di tipo procedure con valore di ritorno ed assegniamogli due argomenti, x e y, che sono i nostri due valori da moltiplicare, come illustrato in **Figura 7**.

A questo punto inseriamo all'interno un blocco moltiplicazione (dal gruppo dei blocchi matematici), collegato al connettore "result" e inseriamo come operatori del blocco matematico le get per i parametri x e y, che possiamo ottenere semplicemente facendo mouse over (portandoci sopra il puntatore del mouse...) sugli argomenti della procedura, come illustrato in **Fig. 8**. Il risultato finale è illustrato in **Fig. 9**.

Un esempio, puramente didattico, di utilizzo della procedura appena realizzata, è riportato in **Fig. 10**.

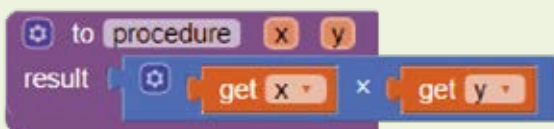


Fig. 10 - Esempio di utilizzo della procedura.

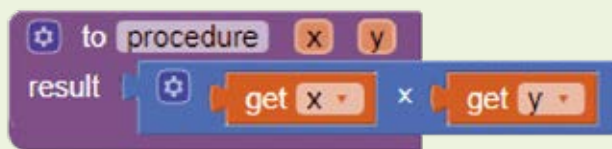
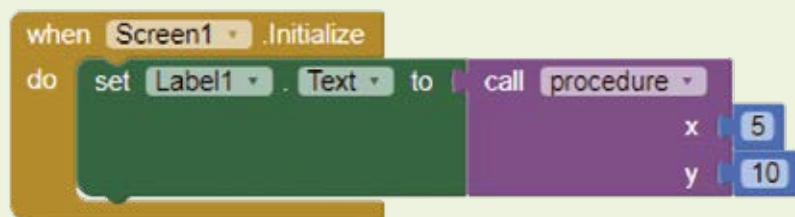


Fig. 9 - Procedura completa.

La famiglia Drawing and Animation

Passiamo adesso all'analisi della famiglia di componenti Drawing and Animation; questa famiglia comprende elementi, come **sprite** e **canvas**, che permettono di realizzare animazioni o anche semplici giochi. In **Fig. 11** è riportato uno screenshot della famiglia Drawing and Animation. Passiamo, come di consueto, all'analisi dei vari componenti, specificando nel dettaglio anche eventi, metodi e proprietà.

Ball

Una semplice sprite di forma rotonda, che può essere posizionata su di un Canvas, dove può reagire a tocchi, swipe e drag, interagire con altre sprites (come altre balls o image sprite) e con gli edge della Canvas nella quale si trova posizionato. Lo sprite ha una serie di proprietà, come velocità e direzione, e si muove in accordo ad esse. Velocità e direzione sono indicate, rispettivamente, in pixel al secondo e in gradi (partendo dal lato destro dello schermo). Ad esempio, per avere una ball che si muova di 8 pixel al secondo, con un intervallo di movimento di 500ms, verso la parte alta dello schermo, si dovrà settare:

- la proprietà speed della ball a 4 pixel;
- la proprietà interval della ball a 500ms;
- la proprietà heading della ball a 90 (0 è il lato destro dello schermo, 90 la parte alta);
- la proprietà enabled a true.



Fig. 11 - Famiglia di componenti Drawing and Animation.

PROPRIETÀ	
SINTASSI	DESCRIZIONE
Enabled	Controlla se lo sprite può o meno muoversi (se la sua velocità è maggiore di 0)
Heading	Controlla l'orientamento dello sprite in gradi. Zero gradi corrisponde al lato destro dello schermo.
Interval	L'intervallo di tempo, in ms, al quale la posizione dello sprite è aggiornata.
Speed	La velocità alla quale lo sprite si muove (in pixel per interval).
X	Coordinata orizzontale dello sprite (si considera l'estremità sinistra come riferimento).
Y	Coordinata verticale dello sprite (si considera l'estremità alta come riferimento).
EVENTI	
SINTASSI	DESCRIZIONE
CollidedWith(component other)	Event handler invocato quando due sprites collidono tra di loro. Viene passato lo sprite con il quale la ball è andata a collidere.
EdgeReached(number edge)	Event handler invocato quando uno sprite raggiunge l'estremità di un canvas. Se in questa circostanza è invocato il metono Bounce passando l'edge come parametro, allora lo sprite rimbalzerà contro l'edge.
Flung (number x, number y, number speed, number heading, number xvel, number yvel)	Event handler invocato quando viene rilevato un swipe veloce sullo schermo. Vengono passati una serie di parametri, tra cui: - Posizione (x,y) di partenza dello swipe, - Speed dello swipe, - Heading dello swipe, - Componenti x e y del vettore velocità dello swipe.
TouchDown(number x, number y)	Evento invocato quando l'utente tocca lo sprite e mantiene il tocco. Vengono fornite le coordinate x e y del tocco.
TouchUp(number x, number y)	Evento invocato quando l'utente interrompe una azione di touch down su uno sprite. Vengono fornite le coordinate x e y del tocco.
Touched(number x, number y)	Evento invocato quando l'utente esegue un tocco veloce su uno sprite. Vengono fornite le coordinate x e y del tocco.
METODI	
SINTASSI	DESCRIZIONE
Bounce(number edge)	Fa rimbalzare uno sprite contro un edge di un canvas. Per ottenere un rimbalzo corretto l'argomento passato deve essere l'edge ritornato dall'evento EdgeReached.
boolean CollidingWith(component other)	Indica se lo sprite si è scontrato con un altro sprite (passato con il parametro other).
MoveTo(number x, number y)	Muove lo sprite alle coordinate x, y.
PointInDirection(number x, number y)	Gira lo sprite in modo da farlo puntare verso le coordinate x,y.
PointTowards(component target)	Muove lo sprite in modo da farlo puntare verso il componente passato come argomento.

Tabella 3 - Eventi, metodi e proprietà del componente Ball.

Analizziamo in dettaglio i più importanti eventi, metodi e proprietà di questo componente, aiutandoci con la **Tabella 3**.

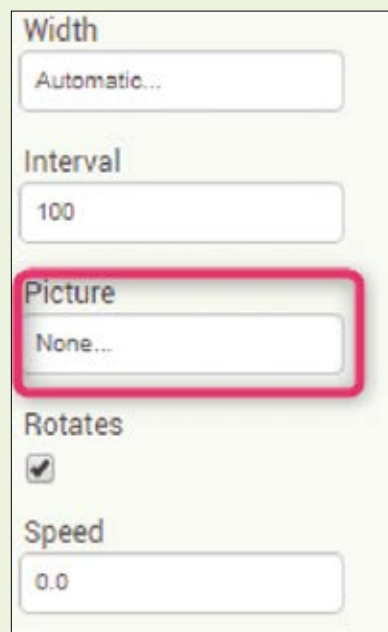
Image Sprite

L'Image Sprite è identico al Ball sprite, con l'unica differenza che prende in input un'immagine, che poi diventerà l'immagine che rappresenta lo sprite sul canvas.

Eventi, metodi e proprietà sono del tutto simili a quelli del ball sprite, con qualche differenza, come la possibilità di importare l'immagine dello sprite, operazione che può essere eseguita dalla component view, come indicato in **Fig. 12**.

Canvas

Il Canvas (Tela) è un pannello rettangolare sensibile al tocco all'interno del quale possono muoversi gli sprites ed è possibile disegnare oggetti. All'interno di un canvas ogni posizione può essere identificata



*Fig. 12
Inserimento
dell'immagine di
uno Image Sprite.*

PROPRIETÀ	
SINTASSI	DESCRIZIONE
BackgroundColor	Colore di Background del canvas
BackgroundImage	Immagine di background del canvas
FontSize	Dimensione dei font del testo scritto sul canvas
LineWidth	Ampiezza delle linee disegnate sul canvas
PaintColor	Colore con cui sono disegnate le linee sul canvas
TextAlignment	Allineamento del testo sul canvas
EVENTI	
SINTASSI	DESCRIZIONE
Flung(number x, number y, number speed, number heading, number xvel, number yvel, boolean flungSprite)	Event handler che intercetta un flung effettuato sul canvas. Vengono passati gli argomenti tipici del flung oltre ad un booleano che indica se la parte iniziale del flung gesture è stata fatta vicino ad uno sprite.
TouchDown(number x, number y)	Evento invocato quando l'utente tocca il canvas e mantiene il tocco. Vengono fornite le coordinate x e y del tocco.
TouchUp(number x, number y)	Evento invocato quando l'utente interrompe una azione di touch down sul canvas. Vengono fornite le coordinate x e y del tocco.
Touched(number x, number y, boolean touchedSprite)	Evento invocato quando l'utente esegue un tocco veloce sul canvas. Vengono fornite le coordinate x e y del tocco.
METODI	
SINTASSI	DESCRIZIONE
Clear()	Cancella tutto quello che è stato disegnato sul canvas, ma mantiene il colore di background ed eventuali immagini di background.
DrawCircle(number x, number y, number r)	Disegna un cerchio alle coordinate x e y, di raggio r (in pixels).
DrawLine(number x1, number y1, number x2, number y2)	Disegna una linea passante per le coordinate fornite come argomenti.
DrawText(text text, number x, number y)	Scrivi un testo alle coordinate x e y.
text Save()	Salva l'immagine del canvas su memoria non volatile.
text SaveAs(text fileName)	Salva l'immagine del canvas su memoria non volatile, nel file chiamato fileName.

Tabella 4 - Eventi, metodi e proprietà per il componente canvas.

per mezzo di due coordinate cartesiane x e y, con le seguenti regole:

- X è il numero di pixel dall'estremità sinistra del canvas;
- Y è il numero di pixel dall'estremità superiore del canvas.

Analizziamo anche per questo componente gli eventi, i metodi e le proprietà, aiutandoci con la **Tabella 4**.

Esempio Pratico con i componenti Drawing and Animation

Passiamo adesso, come di consueto al nostro esempio pratico. Avendo trattato i componenti della famiglia Drawing and Animation, in questa puntata abbiamo deciso di presentare come esempio pratico la realizzazione di una semplice app di gaming. Nello specifico decidiamo di realizzare una sorta di semplice gioco di minigolf, dove bisogna centrare con una pallina una buca posizionata in maniera randomica sulla parte superiore del "campo da

gioco". Il campo sarà un canvas e la palla e la buca due ball sprite, che "tireremo" con uno swipe sullo schermo touch, avvalendoci dell'evento slung. Prevediamo, inoltre, di inserire una label per tenere traccia del punteggio, un componente notifier per la notifica dei punti ed un pulsante per ricominciare il gioco. Inoltre consentiremo alla palla di rimbalzare su tutte le pareti del canvas ed utilizzeremo un componente clock per invalidare un tiro non a segno dopo un timeout di 5 secondi, in modo da evitare rimbalzi indefiniti.

Cominciamo la realizzazione della nostra app d'esempio posizionando gli elementi che ci servono sulla designer view:

- un canvas Cvs_GameArea;
- due Ball Sprite chiamate rispettivamente Spt_Ball ed Spt_Hole;
- un button Btn_Restart;
- due labels chiamate rispettivamente Lbl_ScoreLbl ed Lbl_Score;

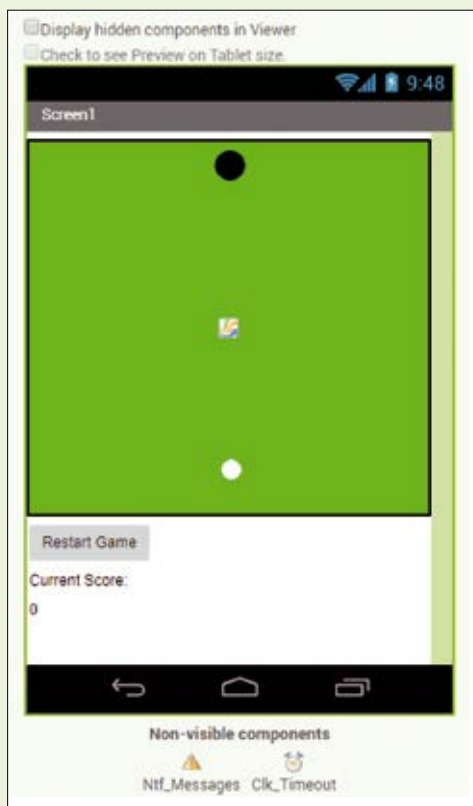


Fig. 13
Layout
dell'app
d'esempio.

- un componente Notifier Ntf_Message;
- un componente Clock Clk_Timeout.

Una volta completato, il layout della nostra app dovrebbe apparire come quello proposto nella Fig. 13. Impostiamo anche le seguenti proprietà statiche:

Canvas:

- Background color: green;
- Height: 70%;
- Width: fill parent.

Spt_Ball:

- Interval: 20;
- Paint Color: White;
- Radius: 8;
- X: 150;
- Y: 300.

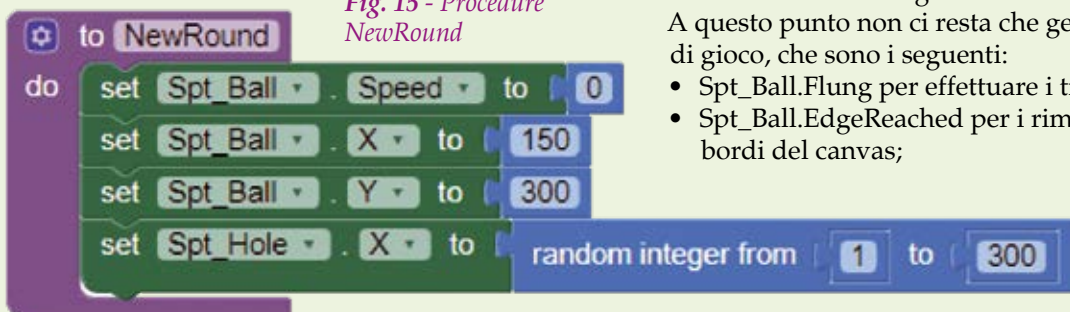


Fig. 15 - Procedure
NewRound

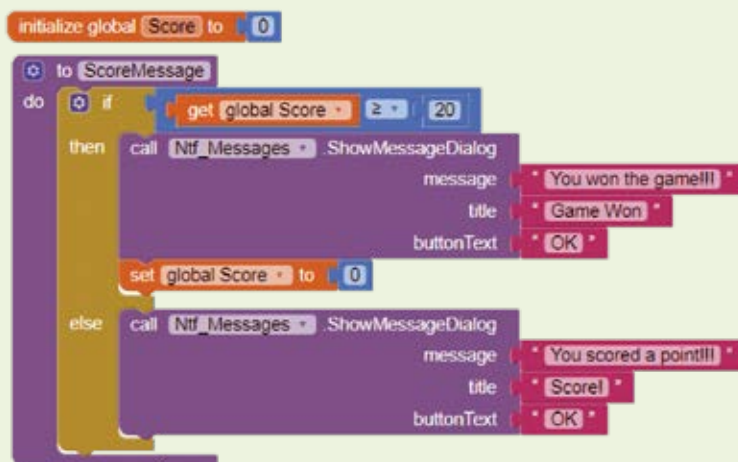


Fig. 14 - Variabile globale e procedure ScoreMessage.

Spt_Hole:

- Paint Color: Black;
- Radius: 12;
- X: 145;
- Y: 20.

Clock:

- Timer Always Fire: false;
- Timer Enabled: false;
- Timer Interval: 5000.

Una volta impostate le proprietà, passiamo sulla block view, dove per prima cosa creiamo una variabile globale Score (inizializzata a zero) e due procedure senza valore di ritorno e senza argomenti, chiamate rispettivamente NewRound e ScoreMessage, le cui implementazioni sono riportate in Fig. 14 e in Fig. 15.

La procedure NewRound ci serve a riposizionare la pallina nella parte centrale bassa del campo e la buca in una nuova posizione random (casuale) sulla parte alta dello schermo.

La procedure ScoreMessage invece ci serve a determinare il messaggio da presentare all'utente (segnatura di un punto o vittoria) tramite il componente notifier. Inoltre, in caso di vittoria (raggiunta se il numero di punti è superiore a 20), viene anche resettata la variabile globale score.

A questo punto non ci resta che gestire i vari eventi di gioco, che sono i seguenti:

- Spt_Ball.Flung per effettuare i tiri;
- Spt_Ball.EdgeReached per i rimbalzi contro i bordi del canvas;

Backpack

Uno degli inconvenienti tipici degli ambienti grafici è la scarsa portabilità del codice tra progetti differenti. Se con gli ambienti testuali, spesso un copia ed incolla è più che sufficiente e abbiamo una grossa varietà di editor che ci aiutano a compiere le più svariate operazioni, con gli ambienti grafici ciò è molto più complesso; spesso l'unico editor valido per elaborare il codice grafico è quello dell'ambiente di sviluppo ed effettuare il copia ed incolla non è sempre agevole, non si può utilizzare il find&replace,

ecc. App Inventor tenta di risolvere questo inconveniente tipico degli ambienti grafici tramite il cosiddetto "Backpack". Il Backpack, ossia lo "zainetto", non è altro che ciò che il suo nome suggerisce: una sorta di zaino virtuale che possiamo stipare con i nostri "snippet" di codice grafico, che può poi essere recuperato successivamente nella stessa o in altre app.

Il backpack si presenta come un'icona a forma di zaino posizionata in alto a destra nella block view. Trascinando del codice grafico al suo interno, l'icona cambia e mostra uno zainetto pieno, come illustrato in **Fig. A**.

Cliccando sul backpack pieno si apre un popup con la lista dei blocchi e degli snippet immagazzinati al suo interno, come illustrato in **Fig. B**. Da questo popup è possibile trascinare i blocchi all'interno del workspace.

Se il backpack è troppo pieno può essere svuotato selezionando l'opzione "empty the backpack" accessibile dal menù contestuale che compare premendo il tasto destro del mouse sul workspace.



Fig. A - Le possibili icone del backpack.

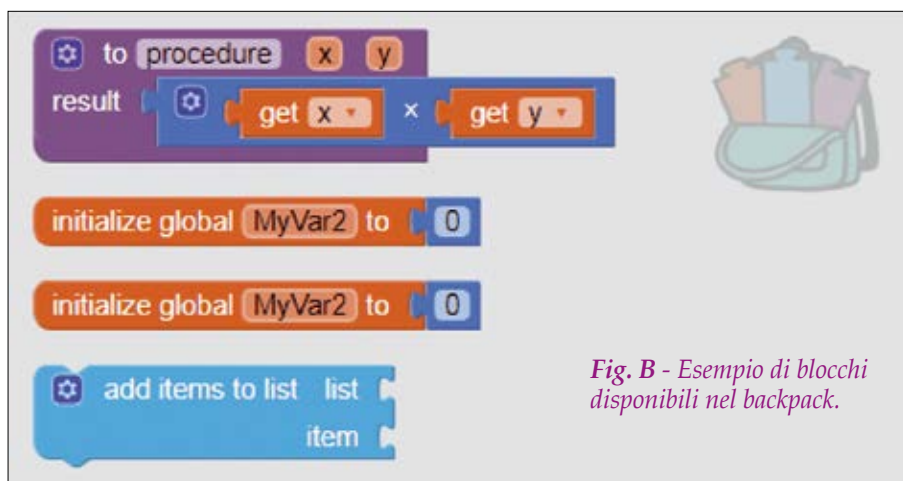


Fig. B - Esempio di blocchi disponibili nel backpack.

- Spt_Ball.CollidedWith per determinare se è stato segnato un punto;
- Btn_Restart.Click per ricominciare un gioco;
- Clk_Timeout.Timer per iniziare un nuovo round allo scadere del timeout (tiro fallito).

Gli snippet di codice grafico che implementano gli eventi elencati sono riportati in **Fig. 16**.

In sostanza, al verificarsi dell'evento di flung viene settata la proprietà speed della pallina al valore dell'argomento speed del flung più una costante

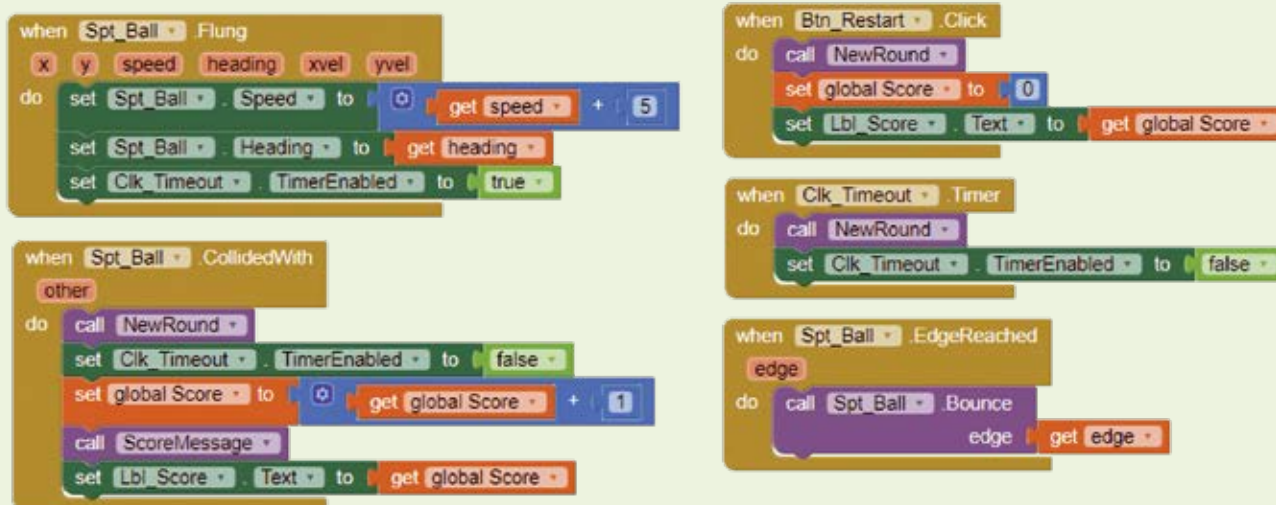


Fig. 16 - Gestione dei vari event handler.

di valore 5 (per rendere il gioco più veloce) e la proprietà heading al valore dell'argomento heading dell'evento. Questo orienta la pallina nella direzione dello swipe. Inoltre viene attivato il timer: da questo momento trascorrono 5 secondi prima che venga triggerato l'evento di timeout. L'evento EdgeReached viene gestito in maniera da generare i rimbalzi, chiamando il metodo Bounce

del componente Spt_Ball e passando l'edge individuato dall'event handler stesso. Al verificarsi dell'evento CollidedWith (pallina in buca) viene chiamata la procedura NewRound, in modo da posizionare gli elementi per un nuovo round di gioco, aggiornato il punteggio e chiamata la procedura ScoreMessage per presentare il messaggio all'utente. Inoltre viene stoppato il timer. Infine con i due eventi BtnRestart.Click e Clk_Timeout.Timer viene iniziato un nuovo round e nel caso della pressione del pulsante azzerato il punteggio (in quanto si tratta di iniziare un nuovo gioco da 0). Potete testare il gioco utilizzando un qualsiasi smartphone android (l'uso dell'emulatore è sconsigliato in questo caso, in quanto non si riuscirebbe ad avere la corretta interazione con l'applicazione, che prevede l'uso fine del touch). Nella Fig. 17 è riportata un'immagine di una fase di gioco, come appare sullo schermo di un dispositivo portatile reale.

Conclusioni

In questa quinta puntata abbiamo analizzato alcuni aspetti nuovi di Appinventor, come la gestione delle stringhe e delle liste, oltre ad aver spiegato come realizzare le procedure. Inoltre abbiamo descritto una nuova famiglia di componenti, la famiglia Drawing and Animation, che risulta fondamentale nello sviluppo di app di gaming. Nella prossima (e conclusiva) puntata del corso vedremo ulteriori aspetti legati alla programmazione ed analizzeremo le ultime famiglie di componenti rimaste, oltre a spiegare come si pubblicano le app realizzate con App Inventor sullo store Google Play per renderle disponibili al pubblico.

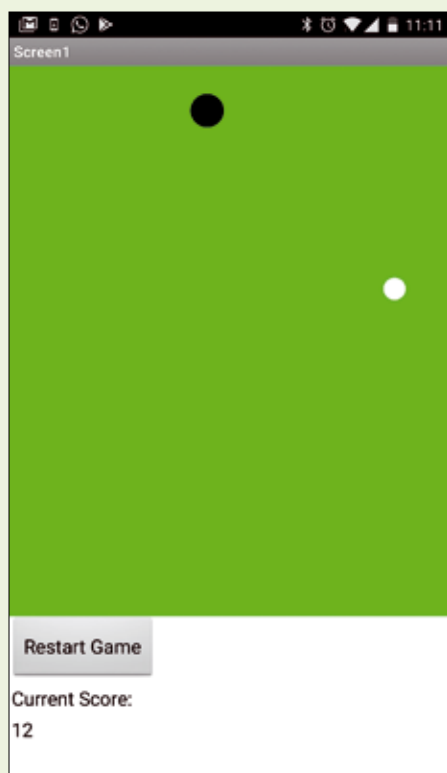


Fig. 17 - Immagine di una fase di gioco.



MIT APP INVENTOR

6

di FRANCESCO FICILI

Continuiamo il nostro viaggio alla scoperta di MIT App Inventor, un tool di sviluppo per applicazioni Android creato dal MIT. In questa sesta e conclusiva puntata analizzeremo le famiglie di componenti maps e connectivity e vedremo come pubblicare le nostre app su google play.

Nella puntata precedente abbiamo proseguito lo studio e l'applicazione pratica dei componenti di MIT Appinventor, analizzando la famiglia di componenti **drawing and animations**, che ci permette di sviluppare applicazioni di gaming sui nostri dispositivi Android. Ricordiamo che i componenti sono gli elementi che permettono di aggiungere funzionalità all'app in fase di sviluppo con App Inventor. Ebbene, in questa puntata concludiamo il discorso e presentiamo due ulteriori famiglie di componenti, ossia **maps** e **connectivity**, e vediamo anche come

fare per pubblicare, sullo store Google Play, le app che abbiamo creato.

La famiglia di componenti Maps

App Inventor dispone della famiglia di componenti Maps, appositamente creati per lo sviluppo di applicazioni di geolocalizzazione e geofencing (applicazioni che sfruttano l'interazione tra una mappa e le informazioni ricavate dal sensore di localizzazione). In Fig. 1 è rappresentata un'immagine dei componenti della famiglia Maps.

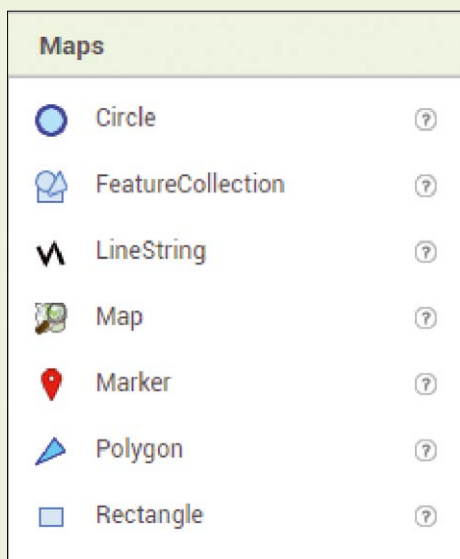


Fig. 1 - Famiglia di componenti Maps.

Non ci soffermeremo molto sulla descrizione di questi componenti in quanto non li utilizzeremo nell'esempio pratico presentato in questa puntata; ci limiteremo piuttosto a fornire una tabella riassuntiva che ne illustra le caratteristiche generali.

COMPONENTE	DESCRIZIONE
Circle	Il circle component permette di visualizzare un cerchio di un dato raggio in metri and una certa latitudine e longitudine, all'interno di una mappa. Questo componente può essere utilizzato per l'implementazione di applicazioni di geofencing, utilizzando i metodi di interazione con il sensore di localizzazione.
FeatureCollection	Il FeatureCollection component raggruppa le feature di una o più mappe insieme.
LineString	Il LineString component permette di disegnare una sequenza continua di linee su una mappa.
Map	Il Map component è un container bidimensionale che permette di renderizzare mappe e piazzare vari tipi di marker al suo interno per identificare punti o zone. Le mappe sono fornite dal servizio OpenStreetMaps.
Marker	Il componente Marker permette di indicare punti in una mappa, come edifici o altri punti di interesse. I marker possono essere customizzati in diversi modi, come ad esempio cambiandone l'aspetto grafico utilizzando un asset grafico dell'app.
Polygon	Il componente Polygon permette di racchiudere un'area di dimensioni arbitrarie su di una mappa. L'uso principale di questo componente è quello di delimitare zone geografiche.
Rectangle	Il componente Rectangle permette di delimitare zone all'interno di un poligono rettangolare, con valori fissi per latitudine e longitudine dei vertici.

Tabella 1 - Componenti della famiglia Maps.

La famiglia di componenti connectivity

La famiglia connectivity di MIT App Inventor comprende questi quattro componenti:

- Activity Starter;
- Bluetooth Client;
- Bluetooth Server;
- Web.

Come potete dedurre dai loro nomi, si tratta principalmente di componenti per la gestione della connettività, in particolare Bluetooth e Web (sia tramite WiFi che tramite rete dati), oltre al componente **Activity Starter**, che permette di lanciare altre applicazioni. In Fig. 2 è rappresentata la famiglia di componenti in oggetto. Analizziamo più in dettaglio i vari componenti di questa famiglia, in modo da comprenderli più nel dettaglio.

Activity Starter

Il componente **Activity Starter** permette di lanciare un activity tramite il metodo start activity.

Una activity in Android può essere vista come una singola schermata (screen) che un utente può vedere sul suo dispositivo in un determinato momento. Sostanzialmente una app è composta da più attivi-

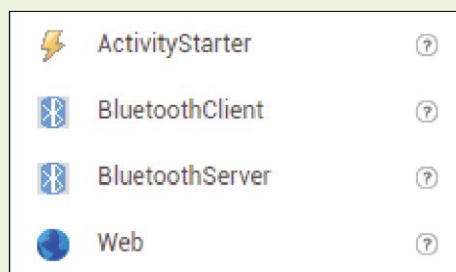


Fig. 2 - Famiglia di componenti connectivity.

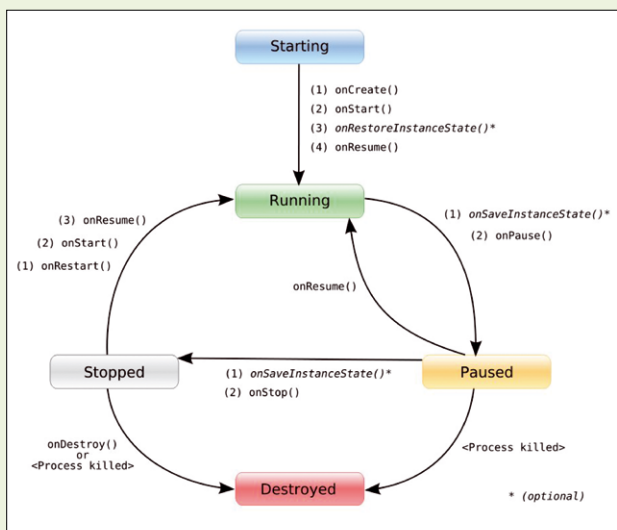


Fig. 3 - Ciclo di vita di una activity Android.

ty, ognuna delle quali può essere vista ed invocata da un'altra app, ed è a questa funzione che il componente **Activity Starter** assolve.

Una volta invocata, una activity ha anche un ciclo di vita, con tutta una serie di eventi correlati.

Il ciclo di vita delle activity in Android è riportato in Fig. 3.

Nella **Tabella 2** sono riportati, come di consueto, eventi, metodi e proprietà del componente.

Tra le activity che possono essere lanciate dal componente activity starter possiamo citare, giusto per fare qualche esempio.

- Avviare l'applicazione camera, cosa che può esse-

re fatto impostando:

- Action: `android.intent.action.MAIN`
- ActivityPackage: `com.android.camera`
- ActivityClass: `com.android.camera.Camera`

- Eseguire una ricerca su web. Assumendo di voler cercare "ElettronicaIn", bisognerebbe impostare:

- Action: `android.intent.action.WEB_SEARCH`
- ExtraKey: `query`
- ExtraValue: `ElettronicaIn`
- ActivityPackage: `com.google.android.providers.enhancedgooglesearch`
- ActivityClass: `com.google.android.providers.enhancedgooglesearch.Launcher`

- Aprire il browser web ad una specifica pagina. Assumendo di voler aprire la pagina web di Elettronica In, bisognerebbe impostare:

- Action: `android.intent.action.VIEW`
- DataUri: `https://www.elettronica.in/`

Bluetooth Client/Server

I moderni smartphone basati su Android sono normalmente dotati di un modulo bluetooth integrato. App Inventor modella il collegamento bluetooth tramite uno schema client/server; esistono quindi due componenti per la gestione del BT, a seconda che si voglia implementare il lato client o il lato server.

Il profilo BT maggiormente supportato è il profilo SPP (Serial Port Profile) che è il profilo base utilizzato dalla maggior parte dei metodi. C'è comunque la possibilità di connettersi tramite un profilo generico, ma il supporto specifico a livello di metodi è molto

PROPRIETÀ	
SINTASSI	DESCRIZIONE
Action	Action della activity.
Activity Class	Class della activity.
Activity Package	Package della activity.
Data Type	DataType della activity.
Data Uri	Data Uri della activity.
Extras	Questa proprietà accetta una lista di coppie key/value che possono poi essere utilizzate nel campo Extras della activity stessa. Sostituisce le proprietà obsolete ExtraKey ed ExtraValue.
EVENTI	
SINTASSI	DESCRIZIONE
AfterActivity(text result)	Evento triggerato quando il metodo StartActivity ritorna.
ActivityCanceled()	Evento triggerato quando il metodo StartActivity ritorna prematuramente, nel caso l'activity sia stata cancellata.
METODI	
SINTASSI	DESCRIZIONE
text ResolveActivity()	Ritorna il nome dell'activity corrispondente all'activityStarter, oppure una stringa vuota se l'activity corrispondente non viene trovata.
StartActivity()	Lancia l'activity corrispondente all'activityStarter.

Tabella 2 - Eventi, metodi e proprietà del componente Activity Starter.

limitato in questo caso.

Nella **Tabella 3** sono riportati eventi, metodi e proprietà del componente BT client.

Invece nella **Tabella 4** sono riportati eventi, metodi e proprietà del componente BT Server.

Web

Il componente **Web** è un elemento di App Inventor non visibile che permette di gestire richieste http GET, POST, PUT e DELETE. Il componente prescinde dal tipo di connessione fisica utilizzata (WiFi o GSM/GPRS), fornendo un'interfaccia uniforme per l'accesso alla rete.

Nella **Tabella 5** sono riportati, come di consueto, eventi, metodi e proprietà del componente.

Esempio Pratico

Passiamo adesso, come di consueto, al nostro esempio pratico, per il quale utilizzeremo il componente BT, creando un semplice terminale BT con profilo SPP (Serial Port Profile).

Come di consueto partiamo dal layout; per realizza-

re il nostro terminale ci occorre:

- un componente BT client, che chiameremo Cmp_BtClient;
- un componente Clock, Cmp_Clock;
- un ListPicker, che chiameremo Lst_DevPicker;
- quattro Label, che chiameremo rispettivamente Lbl_StatusLbl, Lbl_BtSts, Lbl_Receive ed Lbl_ReceiveLbl;
- un pulsante Btn_Send;
- una TextBox Txt_Send.

Il layout dell'app può essere organizzato in tanti modi, in **Fig. 4** ne proponiamo uno.

In questo caso abbiamo posizionato il list picker in alto e questo elemento ci servirà per selezionare il dispositivo BT con il quale intendiamo iniziare una connessione. Sotto al picker abbiamo inserito una label che ci indicherà lo stato della connessione. Più in basso ancora la text box ed il pulsante per l'invio ed infine la label sulla quale stamperemo le stringhe in ricezione. Per quanto riguarda la ricezione configureremo il componente clock per generare un

PROPRIETÀ	
SINTASSI	DESCRIZIONE
AddressesAndNames	Proprietà che contiene indirizzi e nomi dei dispositivi BT accoppiati.
Available	Proprietà che indica se il modulo BT è presente sul dispositivo.
Enabled	Proprietà che indica se il modulo BT è abilitato.
IsConnected	Proprietà che indica se è stata stabilita una connessione.
DelimiterByte	Proprietà che permette di settare il delimiterByte (byte di delimitazione).
EVENTI	
SINTASSI	DESCRIZIONE
Nessuno	
METODI	
SINTASSI	DESCRIZIONE
boolean Connect(text address)	Metodo che permette di effettuare una connessione con il dispositivo il cui indirizzo è passato come parametro, mediante profilo SPP. Ritorna un boolean che indica se la connessione è andata a buon fine.
boolean ConnectWithUUID(text address, text uuid)	Metodo che permette di effettuare una connessione con il dispositivo il cui indirizzo è passato come parametro, mediante un profilo generico (UUID). Ritorna un boolean che indica se la connessione è andata a buon fine.
Disconnect()	Metodo che permette di disconnettersi dal dispositivo BT con il quale si è connessi.
boolean IsDevicePaired(text address)	Metodo che indica se un determinato dispositivo, il cui indirizzo è passato come parametro, è accoppiato.
text ReceiveText(number numberOfBytes)	Metodo che ritorna una stringa di testo ricevuta dal dispositivo connesso, di lunghezza pari al parametro NumberOfBytes. Se il parametro NumberOfBytes è minore di 0 vengono letti tutti i bytes ricevuti prima del carattere di delimitazione.
list ReceiveSignedBytes(number numberOfBytes)	Metodo che ritorna lista di bytes con segno ricevuta dal dispositivo connesso, di lunghezza pari al parametro NumberOfBytes. Se il parametro NumberOfBytes è minore di 0 vengono letti tutti i bytes ricevuti prima del carattere di delimitazione.
list ReceiveUnsignedBytes(number numberOfBytes)	Metodo che ritorna lista di bytes senza segno ricevuta dal dispositivo connesso, di lunghezza pari al parametro NumberOfBytes. Se il parametro NumberOfBytes è minore di 0 vengono letti tutti i bytes ricevuti prima del carattere di delimitazione.
SendBytes(list list)	Metodo che consente di inviare una lista di bytes al dispositivo BT connesso.
SendText(text text)	Metodo che consente di inviare una stringa di caratteri al dispositivo BT connesso.

Tabella 3 - Eventi, metodi e proprietà del componente BT Client.

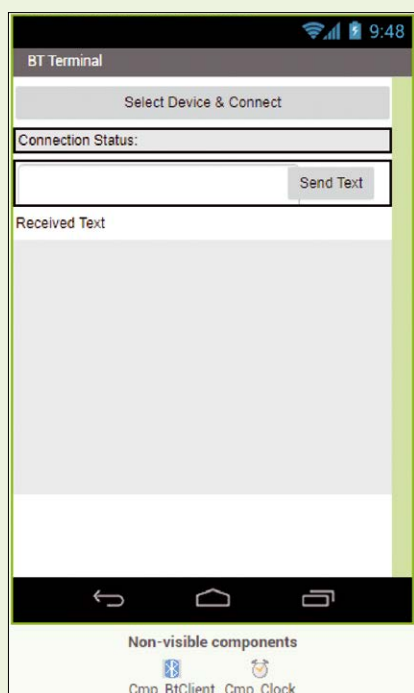


Fig. 4 - Layout dell'applicazione di esempio.

evento ogni 200ms e all'interno dell'evento accoderemo le stringhe ricevute dal BT se ce ne sono. Analizziamo adesso i behaviours, partendo dall'inizializzazione e dalla connessione con il server BT, operazioni rappresentate dai blocchi in Fig. 5. In corrispondenza dell'evento "Initialize" dello screen viene inizializzata la label Lbl_BtSts. Successivamente si popola la lista Lst_DevPicker con indirizzi e nomi dei dispositivi BT associati, prendendoli dalla proprietà `AdresseAndNames` del componente, in corrispondenza dell'evento "BeforePicking". La selezione ottenuta in questa fase (`Lst_DevPicker.Selection`) viene utilizzata in corrispondenza dell'evento "AfterPicking" per connettersi col dispositivo sfruttando il metodo "Connect" del componente BT Client. Sempre in questo evento viene aggiornato il valore della label Lbl_BtStatus a "Connected". La gestione dell'invio e della ricezione dei dati è molto semplice. Per inviare una stringa, in corrispondenza dell'evento "Btn_Send.Click" viene invocato il metodo "SendText" del componente BT Client (viene preventivamente verificato che il componente sia connesso), come illustrato in Fig. 6.

PROPRIETÀ	
SINTASSI	DESCRIZIONE
Available	Proprietà che indica se il modulo BT è presente sul dispositivo.
DelimiterByte	Proprietà che permette di settare il delimiterByte (byte di delimitazione).
Enabled	Proprietà che indica se il modulo BT è abilitato.
IsAccepting	Proprietà che indica se il componente BT server sta accettando una connessione
IsConnected	Proprietà che indica se è stata stabilita una connessione.
EVENTI	
SINTASSI	DESCRIZIONE
ConnectionAccepted()	Evento che segnala che la connessione BT è stata accettata.
METODI	
SINTASSI	DESCRIZIONE
AcceptConnection(text serviceName)	Accetta una connessione via BT tramite profilo SPP.
AcceptConnectionWithUUID(text serviceName, text uuid)	Accetta una connessione via BT tramite profilo generico.
Disconnect()	Metodo che permette di disconnettersi dal dispositivo BT con il quale si è connessi.
number BytesAvailableToReceive()	Metodo che ritorna il numero di bytes che possono essere ricevuti.
list ReceiveSignedBytes (number numberOfBytes)	Metodo che ritorna lista di bytes con segno ricevuta dal dispositivo connesso, di lunghezza pari al parametro NumberOfBytes. Se il parametro NumberOfBytes è minore di 0 vengono letti tutti i bytes ricevuti prima del carattere di delimitazione.
text ReceiveText (number numberOfBytes)	Metodo che ritorna una stringa di testo ricevuta dal dispositivo connesso, di lunghezza pari al parametro NumberOfBytes. Se il parametro NumberOfBytes è minore di 0 vengono letti tutti i bytes ricevuti prima del carattere di delimitazione.
list ReceiveUnsignedBytes (number numberOfBytes)	Metodo che ritorna lista di bytes senza segno ricevuta dal dispositivo connesso, di lunghezza pari al parametro NumberOfBytes. Se il parametro NumberOfBytes è minore di 0 vengono letti tutti i bytes ricevuti prima del carattere di delimitazione.
SendBytes(list list)	Metodo che consente di inviare una lista di bytes al dispositivo BT connesso.
SendText(text text)	Metodo che consente di inviare una stringa di caratteri al dispositivo BT connesso.
StopAccepting()	Metodo per interrompere l'accettazione di una connessione.

Tabella 4 - Eventi, metodi e proprietà del componente BT Server.

Per la ricezione invece si sfrutta il componente clock. Ad ogni occorrenza dell'evento "Time" viene aggiornata la label Lbl_Receive, accodando i caratteri ricevuti sul canale BT mediante l'applicazione del metodo "ReceiveText", come illustrato in Fig. 7. Anche in questo caso si verifica se il componente BT è connesso).

In Fig. 8 è visibile uno screenshot che rappresenta il funzionamento su un dispositivo reale: è stato utilizzato un HC-05 (disponibile sul sito www.futurashop.it/HC05) che effettua un loopback sulla linea seriale.

Pubblicare le app su google play

Passiamo adesso ad una breve guida alla pubblicazione di una applicazione sullo store Google Play. Gli step necessari alla pubblicazione di un'app sono diversi e molti sono opzionali a seconda di quanto dettagliate si vogliano fare le descrizioni, eventuali contenuti aggiuntivi, etc. In questa sede noi ci limiteremo a spiegare i passi minimi indispensabili per la pubblicazione, lasciando al lettore interessato l'esplorazione dei vari step opzionali. Per pubblicare un'app sullo store di Google, il primo step da fare è registrarsi come sviluppatore,

PROPRIETÀ	
SINTASSI	DESCRIZIONE
AllowCookies	Proprietà che indica se i cookies provenienti da una risposta debbano essere salvati ed utilizzati in richieste successive. Il supporto ai cookies è disponibile solo da Android 2.3.
RequestHeaders	Header della richiesta, come lista di due elementi. Il primo elemento rappresenta il field name della richiesta, mentre il secondo rappresenta il valore.
SaveResponse	Proprietà che indica se la risposta debba essere salvata su di un file.
ResponseFileName	Il nome del file nel quale salvare la risposta. Se la proprietà SaveResponse è settata su True e ResponseFileName è vuoto, viene creato un nuovo file con un nome di default.
Url	Url della web request.
EVENTI	
SINTASSI	DESCRIZIONE
GotFile(text url, number responseCode, text responseType, text fileName)	Evento che indica che una richiesta è stata servita e la risposta è stata salvata su un file.
GotText(text url, number responseCode, text responseType, text responseContent)	Evento che indica che una richiesta è stata servita ed il contenuto della risposta viene passato come stringa.
METODI	
SINTASSI	DESCRIZIONE
text BuildRequestData(list list)	Metodo che converte una lista di coppie di elementi nome-valore in una stringa formattata correttamente per l'uso del metodo PostText.
ClearCookies()	Cancella tutti i cookies del web component.
Delete()	Esegue una richiesta http DELETE utilizzando la proprietà url e recupera la risposta. Se la proprietà SaveResponse è settata a true, la risposta verrà salvata nel file indicato e verrà triggerato l'evento GotFile, altrimenti verrà triggerato l'evento GotText.
Get()	Esegue una richiesta http GET utilizzando la proprietà url e recupera la risposta. Se la proprietà SaveResponse è settata a true, la risposta verrà salvata nel file indicato e verrà triggerato l'evento GotFile, altrimenti verrà triggerato l'evento GotText.
PostFile(text path)	Esegue una richiesta http POST utilizzando la proprietà url ed i dati provenienti dal file specificato come parametro. Se la proprietà SaveResponse è settata a true, la risposta verrà salvata nel file indicato e verrà triggerato l'evento GotFile, altrimenti verrà triggerato l'evento GotText.
PostText(text text)	Esegue una richiesta http POST utilizzando la proprietà url e la stringa dati passata come parametro. I caratteri della stringa dati saranno codificati mediante codifica UTF-8. Se la proprietà SaveResponse è settata a true, la risposta verrà salvata nel file indicato e verrà triggerato l'evento GotFile, altrimenti verrà triggerato l'evento GotText.
PutFile(text path)	Esegue una richiesta http PUT utilizzando la proprietà url ed i dati provenienti dal file specificato come parametro. Se la proprietà SaveResponse è settata a true, la risposta verrà salvata nel file indicato e verrà triggerato l'evento GotFile, altrimenti verrà triggerato l'evento GotText.
PutText(text text)	Esegue una richiesta http PUT utilizzando la proprietà url e la stringa dati passata come parametro. I caratteri della stringa dati saranno codificati mediante codifica UTF-8. Se la proprietà SaveResponse è settata a true, la risposta verrà salvata nel file indicato e verrà triggerato l'evento GotFile, altrimenti verrà triggerato l'evento GotText.
any JsonTextDecode(text jsonText)	Decodifica una stringa JSON. Molto utile per la decodifica delle risposte server.
any XMLTextDecode(text XmlText)	Decodifica una stringa XML. Molto utile per la decodifica delle risposte server.

Tabella 5 - Eventi, metodi e proprietà del componente Web.

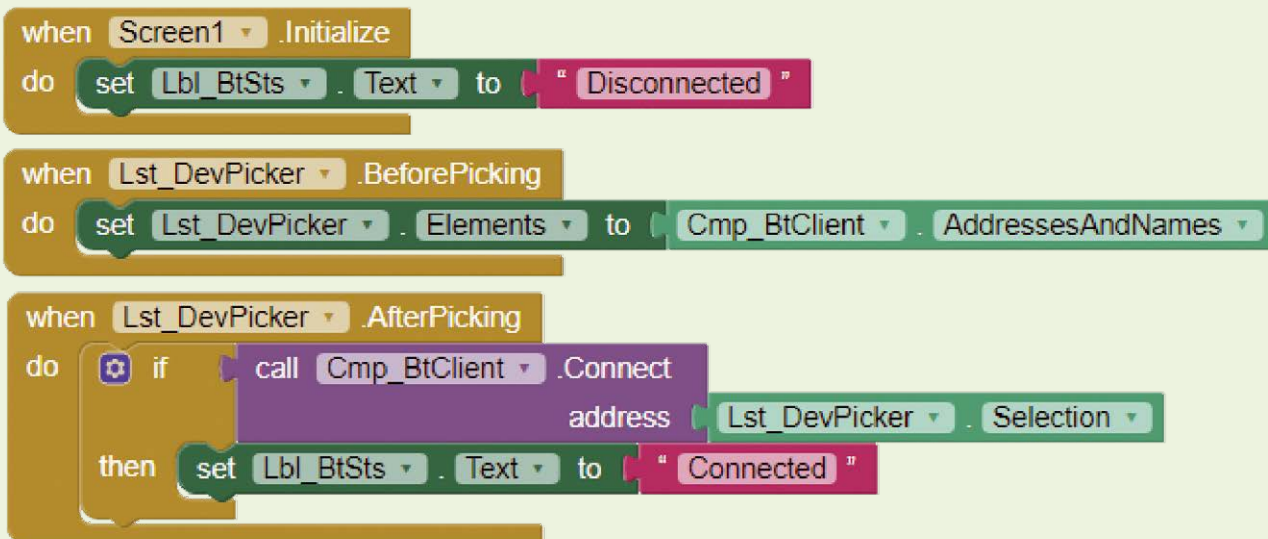


Fig. 5 - Inizializzazione e connessione con il dispositivo.

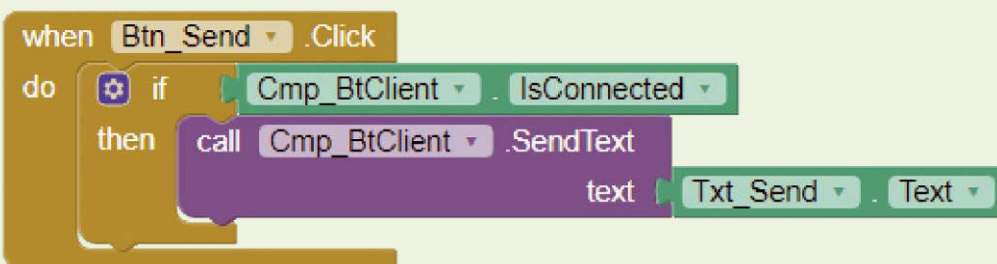


Fig. 6 - Invio dei dati.

cosa che può essere fatta dal seguente link:
<https://play.google.com/apps/publish/signup/>.
 Notate che per consentirvi di effettuare la registrazione Google richiede una fee (un diritto, vale a dire la **Quota di registrazione**) di 25 dollari, quindi è necessario disporre di una carta di credito per effettuare il pagamento mediante il form proposto. La fee è a vita, quindi una volta che l'avrete corrisposta, non dovrete affrontare alcun costo aggiuntivo per ottenere un account da sviluppatore. Una volta ottenuto l'account da sviluppatore potete accedere alla Google Play Console ed iniziare la procedura per la pubblicazione della vostra app. Per iniziare la procedura cliccate sul pulsante "Pubblica un'applicazione Android su Google Play",

come illustrato in Fig. 9.

A questo punto si aprirà la finestra di pop-up di creazione dell'applicazione (Fig. 10) dove dovrete inserire la lingua predefinita ed il titolo, per poi cliccare sul pulsante "crea".

Dopo aver creato l'istanza per la distribuzione, bisogna compilare una per una le sezioni evidenziate in Fig. 11, ossia:

- scheda dello store;
- versioni dell'app;
- classificazione dei contenuti;
- prezzi e distribuzione.

Una volta completata una sezione comparirà l'icona con la spunta di colore verde a fianco.

Tenete in considerazione che tutti i campi che è

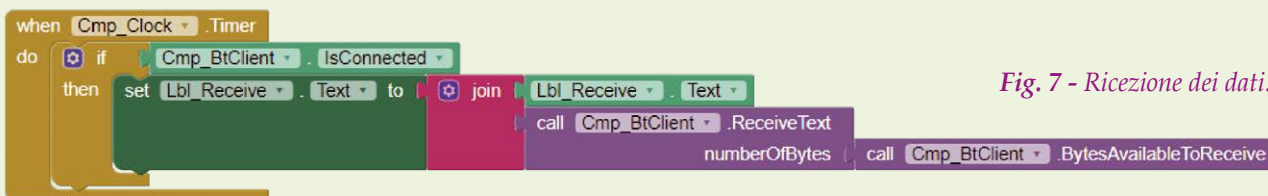


Fig. 7 - Ricezione dei dati.



Fig. 8 - Applicazione di esempio.

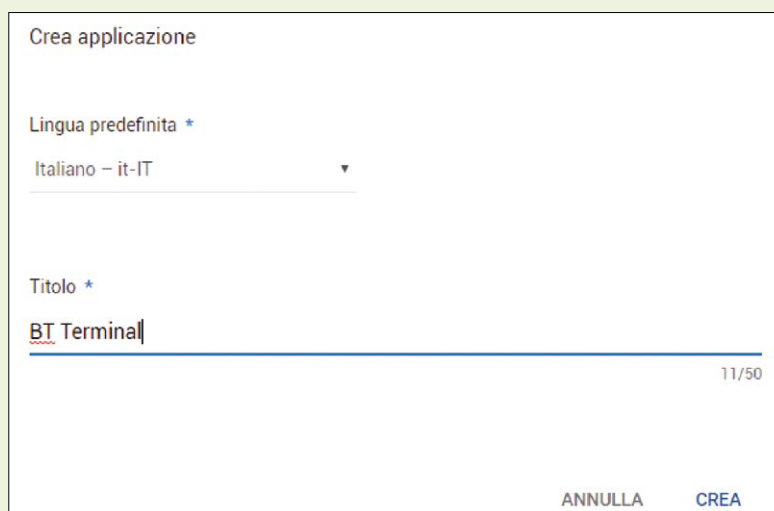


Fig. 10 - Finestra di pop-up di creazione applicazione.



Fig. 9 - Pulsante per avviare la procedura di pubblicazione.

obbligatorio compilare sono contrassegnati con un asterisco.

In Fig. 12 è rappresentato un esempio di compilazione dei campi iniziali della sezione "Scheda dello store".

Una volta riempite le varie sezioni bisogna tornare sulla sezione Versioni dell'App e cliccare sul pulsante blu "Esamina" in basso.

Nella schermata successiva troveremo un breve riepilogo e potremo pubblicare l'app cliccando sul pulsante "Inizia implementazione in versione di produzione", come illustrato in Fig. 13.

A questo punto l'app passerà nello stato "in attesa di pubblicazione" e Google Play vi notificherà non appena la pubblicazione sarà effettiva.

Conclusioni

In questa sesta e conclusiva puntata abbiamo visto le famiglie di componenti maps e connectivity, approfondendo tramite un esempio pratico un com-

ponente di quest'ultima famiglia. Inoltre abbiamo visto come pubblicare le app sullo store ufficiale di Google (Google Play).

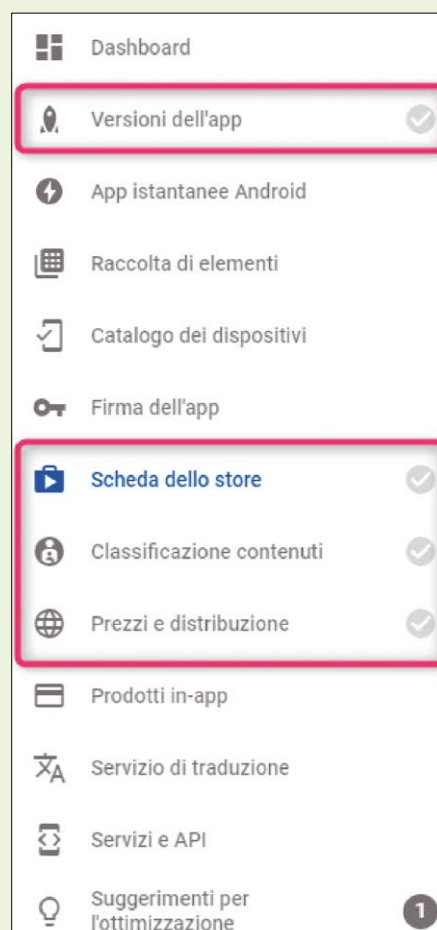


Fig. 11 - Sezioni da compilare per la pubblicazione sullo store.

Dettagli prodotto ITALIANO - IT-IT Gestisci traduzioni ▼

I campi contrassegnati con * devono essere compilati prima della pubblicazione.

Titolo *
Italiano - it-IT BT Terminal 11/50

Descrizione breve *
Italiano - it-IT Simple BT terminal 18/80

Descrizione completa *
Italiano - it-IT This application allows to communicate with a BT server device through SPP profile. 83/4000

Ti invitiamo a leggere le norme relative ai metadati per evitare alcune delle violazioni più comuni relative ai metadati delle app. Assicurati inoltre di rileggere tutte le altre norme del programma prima di inviare le tue app. Se la tua app o la tua scheda dello store sono idonee per i preavvisi al team responsabile dell'esame delle app di Google Play, contattaci prima della pubblicazione.

SALVA BOZZA

Fig. 12 - Campi iniziali della sezione scheda dello store.

Questa puntata completa la nostra panoramica su questo semplice ed efficace strumento di sviluppo per applicazioni Android e getta le basi per l'uti-

lizzo di App Inventor quale complemento di vari dispositivi elettronici gestiti da app e progetti futuri che vi proporremo.

APK in questa versione Espandi tutto

Tipo	Codice versione	Caricato	Dimensioni APK installato	Installazioni su dispositivi attivi
1 APK aggiunto				
▼ APK	1	9 minuti fa	2,94 MB	Nessun dato

Novità di questa versione

Ti consigliamo di aggiungere note per ogni nuova versione. In questo modo gli utenti potranno capire i vantaggi dell'upgrade all'ultima versione dell'app.

[Torna indietro per aggiungere note sulla versione](#)

INDIETRO **ELIMINA** **INIZIA IMPLEMENTAZIONE IN VERSIONE DI PRODUZIONE**

Fig. 13 - Pubblicazione dell'app sullo store.