



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

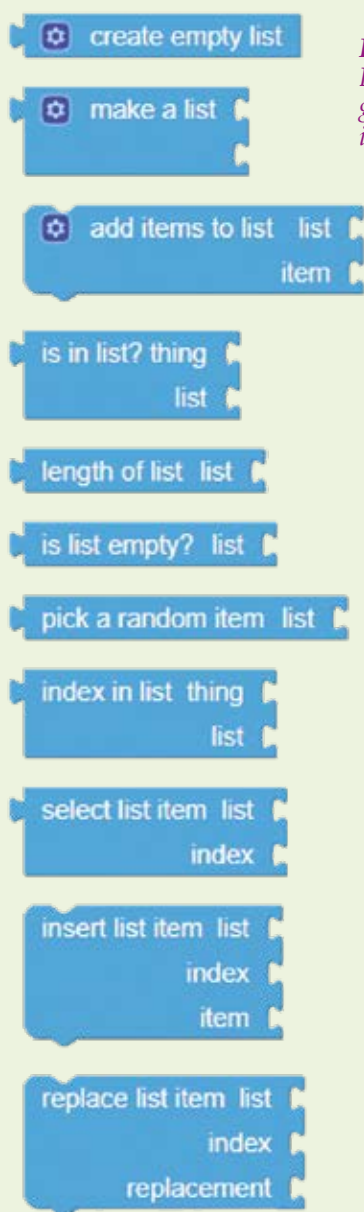
della lista; il primo elemento di una lista ha indice 1, a differenza della stragrande maggioranza dei linguaggi di programmazione, che identifica questo elemento con l'indice 0.

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.*


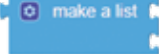
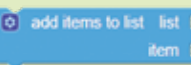
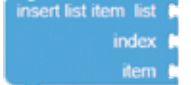
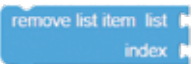

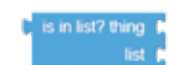
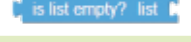
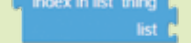
SIMBOLO	OPERAZIONE	DESCRIZIONE
 	Creazione Liste	Questi blocchi consentono di creare una lista vuota o una lista con una serie di elementi.
	Aggiunta di elementi in coda ad una lista	Questo blocco consente di aggiungere nuovi elementi alla lista (è possibile aggiungere elementi cliccando sull'icona a 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.

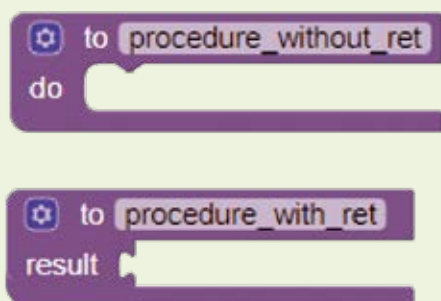


Fig. 3 - Tipi di procedure supportate in AI.

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.

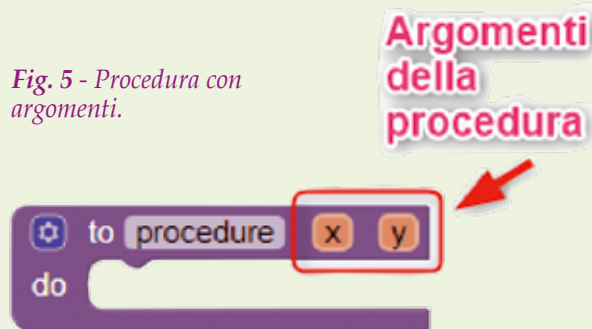
Le procedure in App Inventor

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

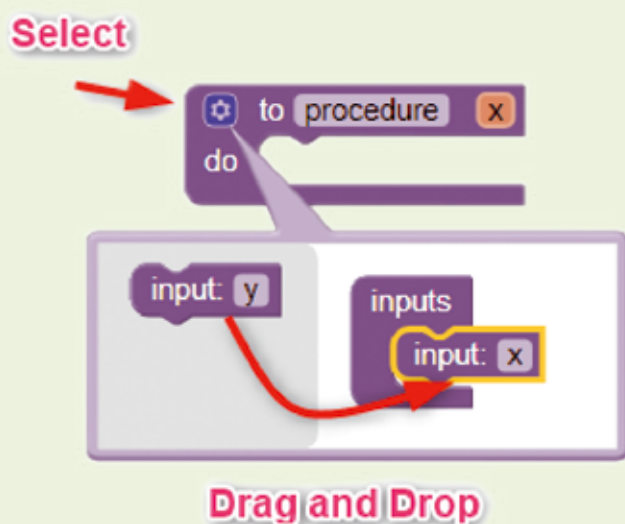


Fig. 4 - Aggiunta degli argomenti ad una procedura.

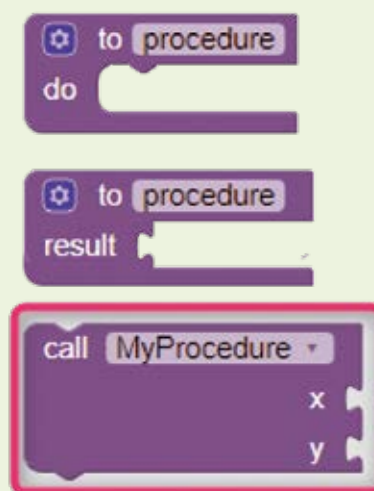


Fig. 6 - Blocco per l'invocazione di una procedura.

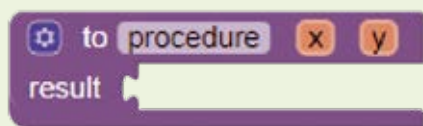


Fig. 7 - Procedura con ritorno ed argomenti x e y.

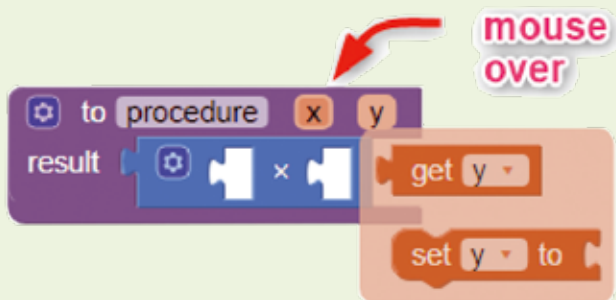


Fig. 8 - Inserimento del blocco moltiplicazione e dei blocchi get degli argomenti della procedura.

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**.

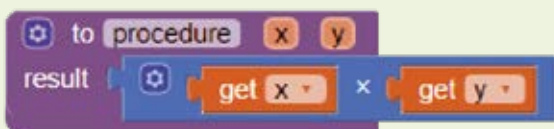


Fig. 10 - Esempio di utilizzo della procedura.

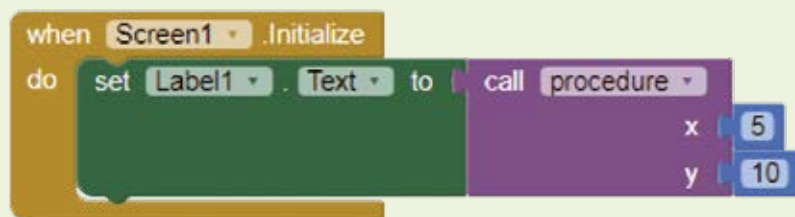


Fig. 11 - Famiglia di componenti Drawing and Animation.

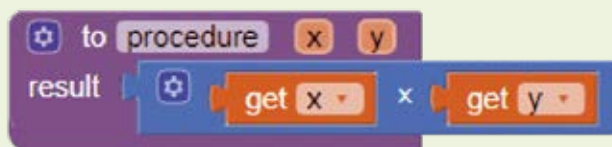


Fig. 9 - Procedura completa.

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.

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**.

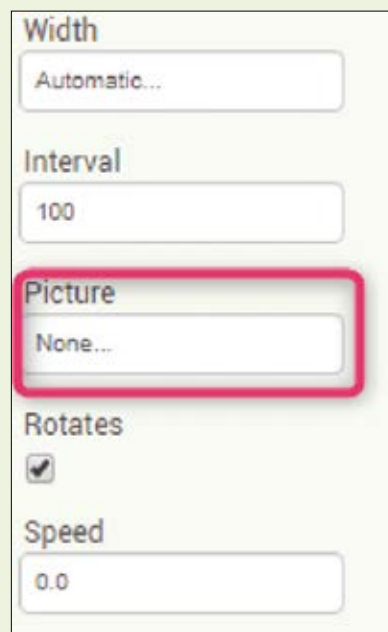
Image Sprite

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)	Scriva 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;

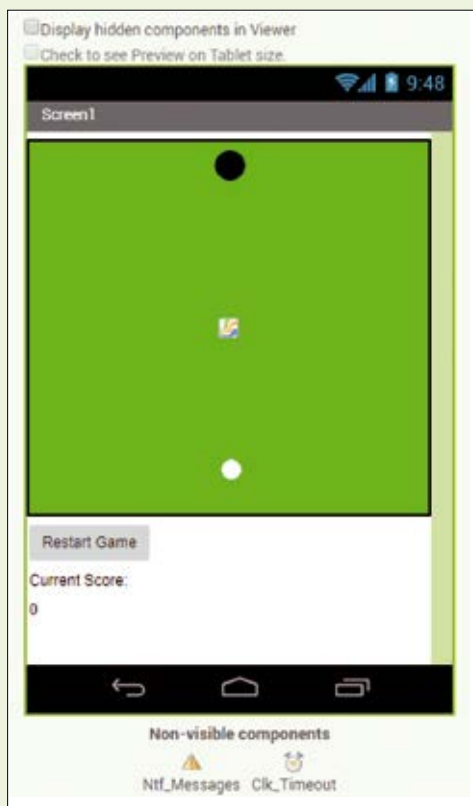


Fig. 13
Layout
dell'app
d'esempio.

- 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.

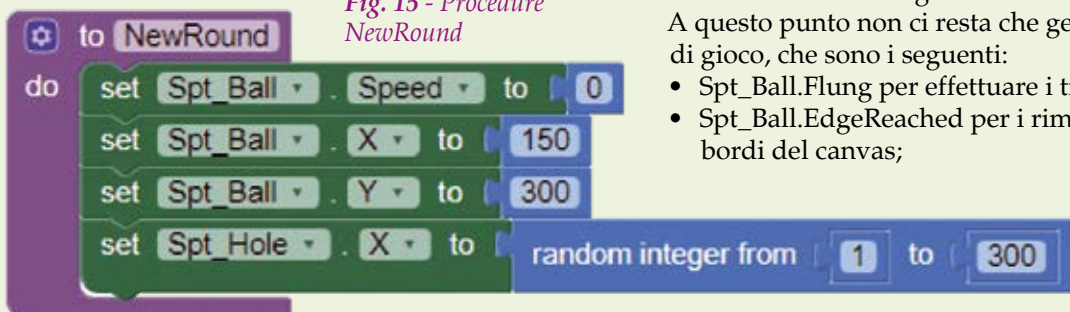


Fig. 15 - Procedure
NewRound

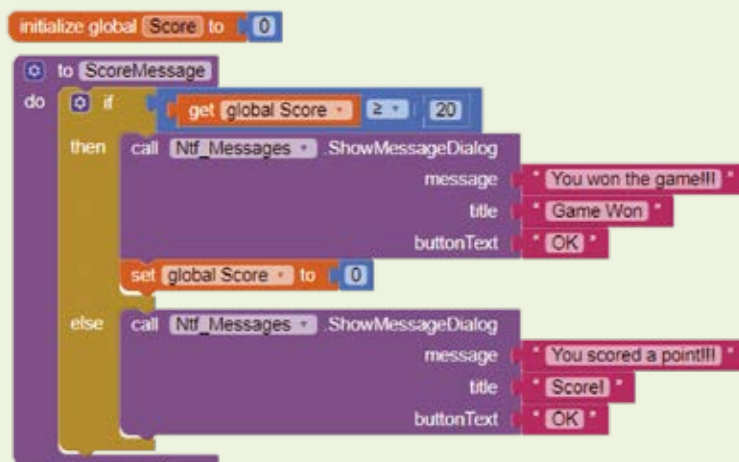


Fig. 14 - Variabile globale e procedure ScoreMessage.

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.

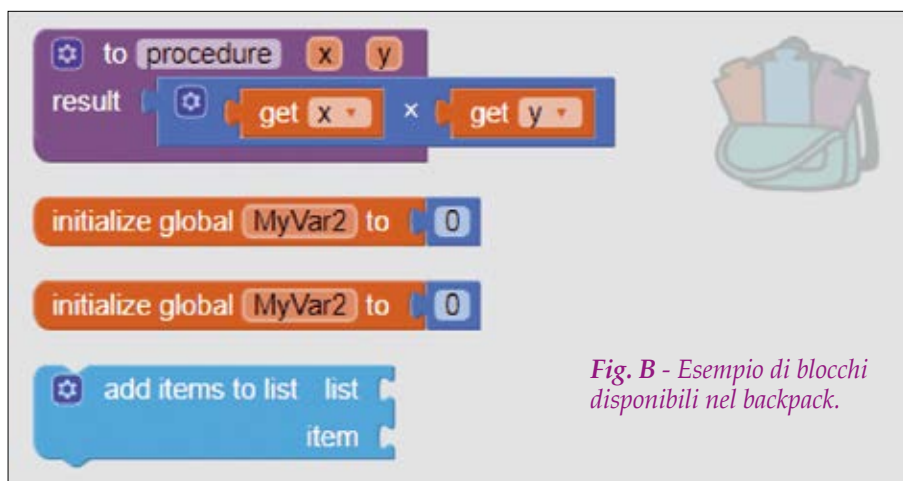


Fig. B - Esempio di blocchi disponibili nel backpack.

- 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

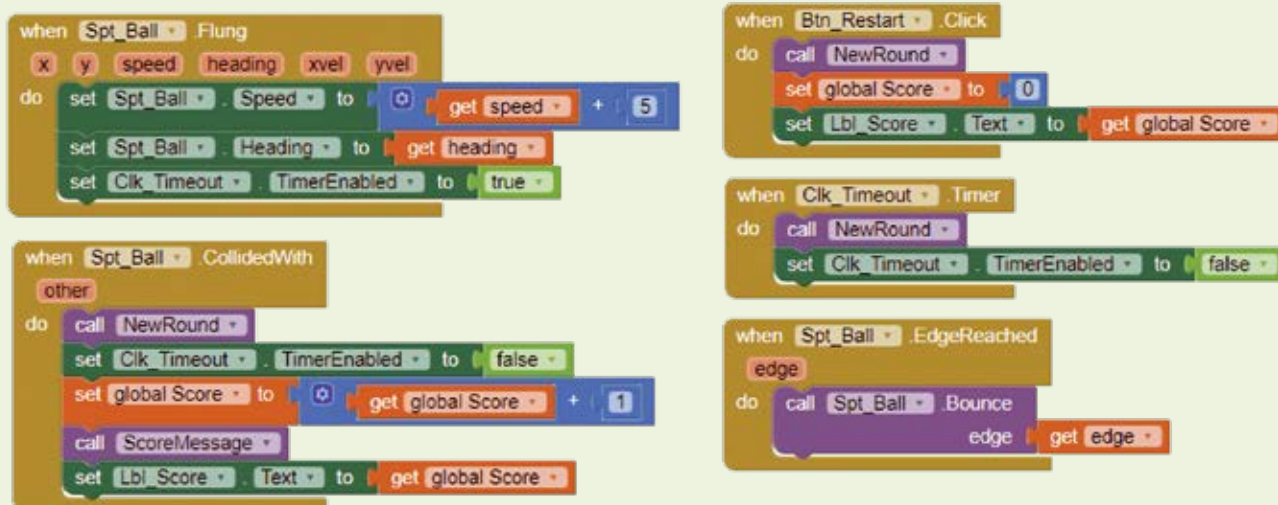


Fig. 16 - Gestione dei vari event handler.

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.

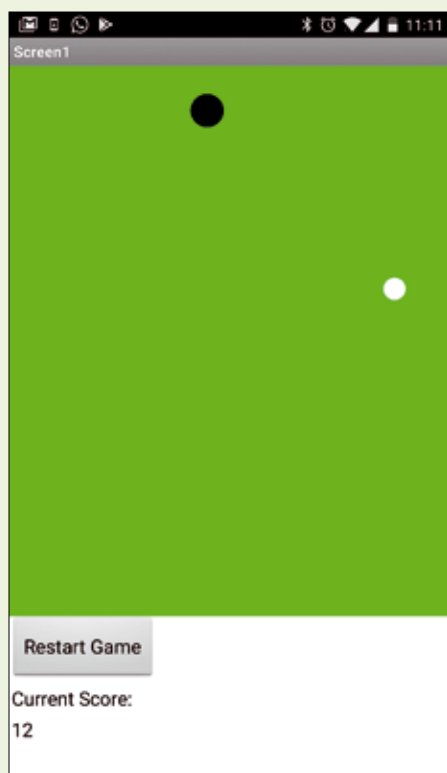


Fig. 17 - Immagine di una fase di gioco.