# Gestire gli eventi di tocco in un ViewGroup

Gestire gli eventi touch in un oggetto ViewGroup richiede particolare attenzione, perché è comune che un oggetto ViewGroup abbia figli che sono obiettivi per eventi di tocco diversi da quelli della ViewGroup stessa. Per assicurarsi che ogni visualizzazione riceva correttamente gli eventi di tocco previsti, eseguire l'override del metodo [onInterceptTouchEvent()](https://developer.android.com/reference/android/view/ViewGroup#onInterceptTouchEvent(android.view.MotionEvent)).

## **Intercettare gli eventi di tocco in un ViewGroup**

Il metodo [onInterceptTouchEvent()](https://developer.android.com/reference/android/view/ViewGroup#onInterceptTouchEvent(android.view.MotionEvent)) viene chiamato ogni volta che viene rilevato un evento touch sulla superficie di un oggetto ViewGrop, anche sulla superficie dei relativi figli. Se restituisce true, l'oggetto [MotionEvent](https://developer.android.com/reference/android/view/MotionEvent) (oggetto utilizzato per segnalare eventi di movimento) viene intercettato, il che significa che non viene passato al figlio, ma piuttosto al metodo [onTouchEvent()](https://developer.android.com/reference/android/view/ViewGroup#onInterceptTouchEvent(android.view.MotionEvent)) dell'elemento padre.

Il metodo [onInterceptTouchEvent()](https://developer.android.com/reference/android/view/ViewGroup#onInterceptTouchEvent(android.view.MotionEvent)) offre a un genitore la possibilità di vedere qualsiasi evento di tocco prima che lo facciano i suoi figli. Se ritorna true da [onInterceptTouchEvent()](https://developer.android.com/reference/android/view/ViewGroup#onInterceptTouchEvent(android.view.MotionEvent)), la visualizzazione figlio che in precedenza gestiva gli eventi di tocco riceve un [ACTION\_CANCEL](https://developer.android.com/reference/android/view/MotionEvent#ACTION_CANCEL) e gli eventi da quel punto in avanti vengono inviati al metodo dell'elemento padre [onTouchEvent()](https://developer.android.com/reference/android/view/View#onTouchEvent(android.view.MotionEvent)) per la gestione abituale. [onInterceptTouchEvent()](https://developer.android.com/reference/android/view/ViewGroup#onInterceptTouchEvent(android.view.MotionEvent)) può anche tornare false e semplicemente spiare gli eventi mentre viaggiano lungo la gerarchia di visualizzazione fino ai loro obiettivi abituali, che gestiranno gli eventi con i propri [onTouchEvent()](https://developer.android.com/reference/android/view/View#onTouchEvent(android.view.MotionEvent)).

Nel frammento seguente, la classe MyViewGroup estende [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup). MyViewGroup contiene più visualizzazioni figlio. Se si trascina il dito su una visualizzazione figlio orizzontalmente, la visualizzazione figlio non deve più ottenere eventi di tocco e MyViewGroup deve gestire gli eventi tocco scorrendone il contenuto. Tuttavia, se si preme un pulsante nella visualizzazione figlio o si scorre la visualizzazione figlio verticalmente, il padre non deve intercettare tali eventi di tocco, perché il figlio è la destinazione prevista. In questi casi [onInterceptTouchEvent()](https://developer.android.com/reference/android/view/ViewGroup#onInterceptTouchEvent(android.view.MotionEvent)), deve restituire false, e il metodo [onTouchEvent()](https://developer.android.com/reference/android/view/View#onTouchEvent(android.view.MotionEvent)) di MyViewGroup non verrà chiamato.

[KOTLIN](https://developer.android.com/training/gestures/viewgroup#kotlin)

class MyViewGroup @JvmOverloads constructor(  
        context: Context,  
        private val mTouchSlop: Int = ViewConfiguration.get(context).scaledTouchSlop  
) : ViewGroup(context) {  
  
    ...  
  
    override fun onInterceptTouchEvent(ev: MotionEvent): Boolean {  
        /\*  
         \* This method JUST determines whether we want to intercept the motion.  
         \* If we return true, onTouchEvent will be called and we do the actual  
         \* scrolling there.  
         \*/  
        return when (ev.actionMasked) {  
            // Always handle the case of the touch gesture being complete.  
            MotionEvent.ACTION\_CANCEL, MotionEvent.ACTION\_UP -> {  
                // Release the scroll.  
                mIsScrolling = false  
                false // Do not intercept touch event, let the child handle it  
            }  
            MotionEvent.ACTION\_MOVE -> {  
                if (mIsScrolling) {  
                    // We're currently scrolling, so yes, intercept the  
                    // touch event!  
                    true  
                } else {  
  
                    // If the user has dragged her finger horizontally more than  
                    // the touch slop, start the scroll  
  
                    // left as an exercise for the reader  
                    val xDiff: Int = calculateDistanceX(ev)  
  
                    // Touch slop should be calculated using ViewConfiguration  
                    // constants.  
                    if (xDiff > mTouchSlop) {  
                        // Start scrolling!  
                        mIsScrolling = true  
                        true  
                    } else {  
                        false  
                    }  
                }  
            }  
            ...  
            else -> {  
                // In general, we don't want to intercept touch events. They should be  
                // handled by the child view.  
                false  
            }  
        }  
    }  
  
    override fun onTouchEvent(event: MotionEvent): Boolean {  
        // Here we actually handle the touch event (e.g. if the action is ACTION\_MOVE,  
        // scroll this container).  
        // This method will only be called if the touch event was intercepted in  
        // onInterceptTouchEvent  
        ...  
    }  
}

Si noti che [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup) fornisce anche un metodo [requestDisallowInterceptTouchEvent()](https://developer.android.com/reference/android/view/ViewGroup#requestDisallowInterceptTouchEvent(boolean)). La [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup) chiama questo metodo quando un figlio non desidera che l'elemento padre e i relativi predecessori intercettino gli eventi di tocco con [onInterceptTouchEvent()](https://developer.android.com/reference/android/view/ViewGroup#onInterceptTouchEvent(android.view.MotionEvent)).

### **Elaborare ACTION\_OUTSIDE eventi**

Se un [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup) riceve un [MotionEvent](https://developer.android.com/reference/android/view/MotionEvent) con un [ACTION\_OUTSIDE](https://developer.android.com/reference/android/view/MotionEvent#ACTION_OUTSIDE), l'evento non verrà inviato ai relativi figli per impostazione predefinita. Per elaborare un [MotionEvent](https://developer.android.com/reference/android/view/MotionEvent) con [ACTION\_OUTSIDE](https://developer.android.com/reference/android/view/MotionEvent#ACTION_OUTSIDE), eseguire l'override del metodo [dispatchTouchEvent(MotionEvent event)](https://developer.android.com/reference/android/view/ViewGroup#dispatchTouchEvent(android.view.MotionEvent)) per eseguire l'invio alla [View](https://developer.android.com/reference/android/view/View) appropriata o gestirlo nella [Window.Callback](https://developer.android.com/reference/android/view/Window.Callback) pertinente (ad esempio [Activity](https://developer.android.com/reference/android/app/Activity)).

## **Utilizzare le costanti ViewConfiguration**

Il frammento precedente utilizza la corrente [ViewConfiguration](https://developer.android.com/reference/android/view/ViewConfiguration) per inizializzare una variabile denominata mTouchSlop. Puoi utilizzare la classe [ViewConfiguration](https://developer.android.com/reference/android/view/ViewConfiguration) per accedere a distanze, velocità e tempi comuni utilizzati dal sistema Android.

"Touch slop" si riferisce alla distanza in pixel che il tocco di un utente può vagare prima che il gesto sia interpretato come scorrimento. Touch slop viene in genere utilizzato per impedire lo scorrimento accidentale quando l'utente esegue altre operazioni di tocco, ad esempio il tocco di elementi sullo schermo.

Altri due metodi di [ViewConfiguration](https://developer.android.com/reference/android/view/ViewConfiguration) comunemente utilizzati sono [getScaledMinimumFlingVelocity()](https://developer.android.com/reference/android/view/ViewConfiguration#getScaledMinimumFlingVelocity()) e [getScaledMaximumFlingVelocity()](https://developer.android.com/reference/android/view/ViewConfiguration#getScaledMaximumFlingVelocity()). Questi metodi restituiscono la velocità minima e massima (rispettivamente) per iniziare un gesto, misurata in pixel al secondo. Per esempio:

[KOTLIN](https://developer.android.com/training/gestures/viewgroup#kotlin)

private val vc: ViewConfiguration = ViewConfiguration.get(context)  
private val mSlop: Int = vc.scaledTouchSlop  
private val mMinFlingVelocity: Int = vc.scaledMinimumFlingVelocity  
private val mMaxFlingVelocity: Int = vc.scaledMaximumFlingVelocity  
  
...  
  
MotionEvent.ACTION\_MOVE -> {  
    ...  
    val deltaX: Float = motionEvent.rawX - mDownX  
    if (Math.abs(deltaX) > mSlop) {  
        // A swipe occurred, do something  
    }  
    return false  
}  
  
...  
  
MotionEvent.ACTION\_UP -> {  
    ...  
    if (velocityX in mMinFlingVelocity..mMaxFlingVelocity && velocityY < velocityX) {  
        // The criteria have been satisfied, do something  
    }  
}

## **Estendere l'area toccabile di una visualizzazione figlio**

Android fornisce la classe [TouchDelegate](https://developer.android.com/reference/android/view/TouchDelegate) per consentire a un genitore di estendere l'area toccabile di una vista figlio oltre i limiti del bambino. Questo è utile quando il bambino deve essere piccolo, ma dovrebbe avere una regione di tocco più grande. È inoltre possibile utilizzare questo approccio per ridurre l'area di tocco del bambino, se necessario.

Nell'esempio seguente, un [ImageButton](https://developer.android.com/reference/android/widget/ImageButton) è la "delegate view” (cioè il figlio la cui area di tocco estenderà quella dell'elemento padre). Ecco il file di layout:

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
     android:id="@+id/parent\_layout"  
     android:layout\_width="match\_parent"  
     android:layout\_height="match\_parent"  
     tools:context=".MainActivity" >  
  
     <ImageButton android:id="@+id/button"  
          android:layout\_width="wrap\_content"  
          android:layout\_height="wrap\_content"  
          android:background="@null"  
          android:src="@drawable/icon" />  
</RelativeLayout>

Lo snippet riportato di seguito fa quanto segue:

* Ottiene la visualizzazione padre e invia un [Runnable](https://developer.android.com/reference/java/lang/Runnable)[getHitRect()](https://developer.android.com/reference/android/view/View#getHitRect(android.graphics.Rect)) nel thread dell'interfaccia utente. Ciò garantisce che il genitore dispone i propri figli prima di chiamare il metodo [getHitRect()](https://developer.android.com/reference/android/view/View#getHitRect(android.graphics.Rect)). Il metodo [getHitRect()](https://developer.android.com/reference/android/view/View#getHitRect(android.graphics.Rect)) ottiene il rettangolo di hit del bambino (area toccabile) nelle coordinate dell'elemento padre.
* Trova la visualizzazione figlio [ImageButton](https://developer.android.com/reference/android/widget/ImageButton) e chiama [getHitRect()](https://developer.android.com/reference/android/view/View#getHitRect(android.graphics.Rect)) per ottenere i limiti dell'area toccabile del bambino.
* Estende i limiti del rettangolo di hit dell'oggetto [ImageButton](https://developer.android.com/reference/android/widget/ImageButton).
* Crea un'istanza [TouchDelegate](https://developer.android.com/reference/android/view/TouchDelegate) passando il rettangolo di hit espanso e la visualizzazione figlio [ImageButton](https://developer.android.com/reference/android/widget/ImageButton) come parametri.
* Imposta [TouchDelegate](https://developer.android.com/reference/android/view/TouchDelegate) nella visualizzazione padre, in modo che i tocchi all'interno dei limiti del delegato tocco siano instradati al figlio.

In qualità di delegato tocco per la visualizzazione figlio [ImageButton](https://developer.android.com/reference/android/widget/ImageButton), la visualizzazione padre riceverà tutti gli eventi di tocco. Se l'evento di tocco si è verificato all'interno del rettangolo di hit del bambino, l'elemento padre passerà l'evento tocco al figlio per la gestione.

[KOTLIN](https://developer.android.com/training/gestures/viewgroup#kotlin)

public class MainActivity : Activity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity\_main)  
  
        // Post in the parent's message queue to make sure the parent  
        // lays out its children before you call getHitRect()  
        findViewById<View>(R.id.parent\_layout).post {  
            // The bounds for the delegate view (an ImageButton  
            // in this example)  
            val delegateArea = Rect()  
            val myButton = findViewById<ImageButton>(R.id.button).apply {  
                isEnabled = true  
                setOnClickListener {  
                    Toast.makeText(  
                            this@MainActivity,  
                            "Touch occurred within ImageButton touch region.",  
                            Toast.LENGTH\_SHORT  
                    ).show()  
                }  
  
                // The hit rectangle for the ImageButton  
                getHitRect(delegateArea)  
            }  
  
            // Extend the touch area of the ImageButton beyond its bounds  
            // on the right and bottom.  
            delegateArea.right += 100  
            delegateArea.bottom += 100  
  
            // Sets the TouchDelegate on the parent view, such that touches  
            // within the touch delegate bounds are routed to the child.  
            (myButton.parent as? View)?.apply {  
                // Instantiate a TouchDelegate.  
                // "delegateArea" is the bounds in local coordinates of  
                // the containing view to be mapped to the delegate view.  
                // "myButton" is the child view that should receive motion  
                // events.  
                touchDelegate = TouchDelegate(delegateArea, myButton)  
            }  
        }  
    }  
}

# Gestire l'input da tastiera

Il sistema Android mostra una tastiera su schermo, nota come metodo di input soft quando un campo di testo nell'interfaccia utente riceve lo stato attivo. Per offrire la migliore esperienza utente, è possibile specificare le caratteristiche sul tipo di input previsto (ad esempio se si tratta di un numero di telefono o di un indirizzo di posta elettronica) e sul comportamento del metodo di input, ad esempio se esegue la correzione automatica per gli errori di ortografia.

Oltre ai metodi di input su schermo, Android supporta anche le tastiere hardware, quindi è importante che l'app ottimizzi la propria esperienza utente per l'interazione che potrebbe verificarsi tramite una tastiera collegata.

# Specificare il tipo di metodo di input

Ogni campo di testo prevede un determinato tipo di input di testo, ad esempio un indirizzo e-mail, un numero di telefono o solo testo normale. Pertanto, è importante specificare il tipo di input per ogni campo di testo nell'app in modo che il sistema 2000 il metodo di input soft appropriato (ad esempio una tastiera su schermo).

Oltre al tipo di pulsanti disponibili con un metodo di input, è necessario specificare comportamenti quali se il metodo di input fornisce suggerimenti ortografici, capitalizza nuove frasi e sostituisce il pulsante di ritorno a capo con un pulsante di azione, ad esempio **Fatto** o **Successivo**. Questa lezione mostra come specificare queste caratteristiche.

## **Specificare il tipo di tastiera**

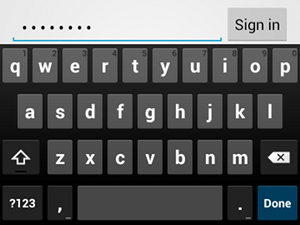
È sempre necessario dichiarare il metodo di input per i campi di testo aggiungendo [l'attributo android:inputType](https://developer.android.com/reference/android/widget/TextView#attr_android:inputType)all'elemento [<EditText>](https://developer.android.com/reference/android/widget/EditText).



**Figura 1.** Tipo di input.phone

Ad esempio, se si desidera un metodo di input per l'immissione di un numero di telefono, utilizzare il valore "phone":

<EditText  
    android:id="@+id/phone"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="wrap\_content"  
    android:hint="@string/phone\_hint"  
    android:inputType="phone" />



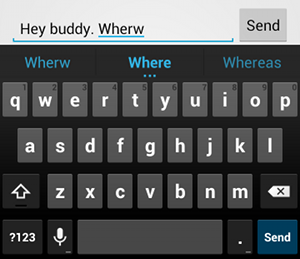
**Figura 2.** Tipo di input.textPassword

Oppure, se il campo di testo è per una password, usa il valore "textPassword" in modo che il campo di testo nasconda l'input dell'utente:

<EditText  
    android:id="@+id/password"  
    android:hint="@string/password\_hint"  
    android:inputType="textPassword"  
    ... />

Esistono diversi valori possibili documentati con [l'attributo android:inputType](https://developer.android.com/reference/android/widget/TextView#attr_android:inputType)e alcuni dei valori possono essere combinati per specificare l'aspetto del metodo di input e comportamenti aggiuntivi.

## Abilitare suggerimenti ortografici e altri comportamenti



**Figura 3.** L'aggiunta fornisce la correzione automatica per gli errori di ortografia.textAutoCorrect

[L'attributo android:inputType](https://developer.android.com/reference/android/widget/TextView#attr_android:inputType)consente di specificare vari comportamenti per il metodo di input. Ancora più importante, se il campo di testo è destinato all'input di testo di base (ad esempio per un messaggio di testo), è necessario abilitare la correzione ortografica automatica con il valore."textAutoCorrect"

È possibile combinare comportamenti e stili di metodo di input diversi con l['attributo android:inputType.](https://developer.android.com/reference/android/widget/TextView#attr_android:inputType) Ad esempio, ecco come creare un campo di testo che capitalizzi la prima parola di una frase e corregga automaticamente gli errori di ortografia:

<EditText  
    android:id="@+id/message"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:inputType=  
        "textCapSentences|textAutoCorrect"  
    ... />

## **Specificare l'azione del metodo di input**

La maggior parte dei metodi di input soft fornisce un pulsante di azione utente nell'angolo inferiore appropriato per il campo di testo corrente. Per impostazione predefinita, il sistema utilizza questo pulsante per un'azione **Avanti** o **Fatto,** a meno che il campo di testo non consenta il testo su più linee (ad esempio con android:inputType="textMultiLine"), in questo caso il pulsante di azione è un ritorno a capo. Tuttavia, è possibile specificare azioni aggiuntive che potrebbero essere più appropriate per il campo di testo, ad esempio **Invia** o **Vai**.

Per specificare il pulsante di azione della tastiera, utilizzare [l'attributo android:imeOptions](https://developer.android.com/reference/android/widget/TextView#attr_android:imeOptions)con un valore di azione quale "actionSend" o "actionSearch". Per esempio:



**Figura 4.** Il pulsante Invia viene visualizzato quando si dichiara android:imeOptions="actionSend".

<EditText  
    android:id="@+id/search"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="wrap\_content"  
    android:hint="@string/search\_hint"  
    android:inputType="text"  
    android:imeOptions="actionSend" />

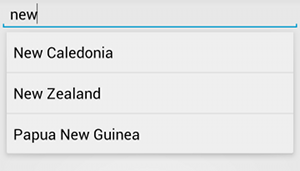
È quindi possibile ascoltare le pressioni sul pulsante di azione definendo un [TextView.OnEditorActionListener](https://developer.android.com/reference/android/widget/TextView.OnEditorActionListener) per l'elemento [EditText](https://developer.android.com/reference/android/widget/EditText). Nel listener rispondere all'ID azione IME appropriato definito nella classe [EditorInfo](https://developer.android.com/reference/android/view/inputmethod/EditorInfo), ad esempio [IME\_ACTION\_SEND](https://developer.android.com/reference/android/view/inputmethod/EditorInfo#IME_ACTION_SEND). Per esempio:

[KOTLIN](https://developer.android.com/training/keyboard-input/style#kotlin)

findViewById<EditText>(R.id.search).setOnEditorActionListener { v, actionId, event ->  
    return@setOnEditorActionListener when (actionId) {  
        EditorInfo.IME\_ACTION\_SEND -> {  
            sendMessage()  
            true  
        }  
        else -> false  
    }  
}

## **Fornire suggerimenti per il completamento automatico**

Se si desidera fornire suggerimenti agli utenti mentre digitano, è possibile utilizzare una sottoclasse di [EditText](https://developer.android.com/reference/android/widget/EditText) denominata [AutoCompleteTextView](https://developer.android.com/reference/android/widget/AutoCompleteTextView). Per implementare il completamento automatico, è necessario specificare un [Adapter](https://developer.android.com/reference/android/widget/Adapter) che fornisca i suggerimenti di testo. Sono disponibili diversi tipi di adapter, a seconda della posizione da cui provengono i dati, ad esempio da un database o da un array.



**Figura 5.** Esempio di [AutoCompleteTextView](https://developer.android.com/reference/android/widget/AutoCompleteTextView) con suggerimenti di testo.

Nella procedura seguente viene descritto come impostare un [AutoCompleteTextView](https://developer.android.com/reference/android/widget/AutoCompleteTextView) che fornisce suggerimenti da una matrice, utilizzando [ArrayAdapter](https://developer.android.com/reference/android/widget/ArrayAdapter):

1. Aggiungere  [AutoCompleteTextView](https://developer.android.com/reference/android/widget/AutoCompleteTextView) al layout. Ecco un layout con solo il campo di testo:

<?xml version="1.0" encoding="utf-8"?>  
<AutoCompleteTextView xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/autocomplete\_country"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="wrap\_content" />

1. Definire la matrice che contiene tutti i suggerimenti di testo. Ad esempio, ecco una matrice di nomi di paese definiti in un file di risorse XML (res/values/strings.xml):

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string-array name="countries\_array">  
        <item>Afghanistan</item>  
        <item>Albania</item>  
        <item>Algeria</item>  
        <item>American Samoa</item>  
        <item>Andorra</item>  
        <item>Angola</item>  
        <item>Anguilla</item>  
        <item>Antarctica</item>  
        ...  
    </string-array>  
</resources>

1. In [Activity](https://developer.android.com/reference/android/app/Activity) o [Fragment](https://developer.android.com/reference/android/app/Fragment) utilizzare il codice seguente per specificare la scheda che fornisce i suggerimenti:

[KOTLIN](https://developer.android.com/training/keyboard-input/style#kotlin)

// Get a reference to the AutoCompleteTextView in the layout  
val textView = findViewById(R.id.autocomplete\_country) as AutoCompleteTextView  
// Get the string array  
val countries: Array<out String> = resources.getStringArray(R.array.countries\_array)  
// Create the adapter and set it to the AutoCompleteTextView  
ArrayAdapter<String>(this, android.R.layout.simple\_list\_item\_1, countries).also { adapter ->  
    textView.setAdapter(adapter)  
}

Qui viene inizializzato un nuovo [ArrayAdapter](https://developer.android.com/reference/android/widget/ArrayAdapter) per associare ogni elemento dell’array di stringhe countries\_array a un [TextView](https://developer.android.com/reference/android/widget/TextView) presente nel layout simple\_list\_item\_1 (si tratta di un layout fornito da Android che fornisce un aspetto standard per il testo in un elenco).

1. Assegnare l'adattatore all’[AutoCompleteTextView](https://developer.android.com/reference/android/widget/AutoCompleteTextView) chiamando [setAdapter()](https://developer.android.com/reference/android/widget/AutoCompleteTextView" \l "setAdapter(T)).

# Gestire la visibilità del metodo di input

Quando lo stato attivo dell'input si sposta all'interno o all'uscita da un campo di testo modificabile, Android mostra o nasconde il metodo di input (ad esempio la tastiera su schermo) a seconda dei casi. Il sistema prende anche decisioni su come appaiono l'interfaccia utente e il campo di testo sopra il metodo di input. Ad esempio, quando lo spazio verticale sullo schermo è vincolato, il campo di testo potrebbe riempire tutto lo spazio sopra il metodo di input. Per la maggior parte delle app, questi comportamenti predefiniti sono tutto ciò che serve.

In alcuni casi, tuttavia, potresti voler controllare più direttamente la visibilità del metodo di input e specificare come vuoi che il tuo layout venga visualizzato quando il metodo di input è visibile. Questa lezione spiega come controllare e rispondere alla visibilità del metodo di input.

## **Mostrare il metodo di input all'avvio dell'attività**

Sebbene Android dia fuoco al primo campo di testo nel layout all'avvio dell'attività, non mostra il metodo di input. Questo comportamento è appropriato perché l'immissione di testo potrebbe non essere l'attività principale nell'attività. Tuttavia, se l'immissione di testo è effettivamente l'attività principale (ad esempio in una schermata di accesso), probabilmente si desidera che il metodo di input venga visualizzato per impostazione predefinita.

Per visualizzare il metodo di input all'avvio dell'attività, aggiungere [l'attributo android:windowSoftInputMode](https://developer.android.com/guide/topics/manifest/activity-element#wsoft)all'elemento <activity> con il valore "stateVisible". Per esempio:

<application ... >  
    <activity  
        android:windowSoftInputMode="stateVisible" ... >  
        ...  
    </activity>  
    ...  
</application>

**Nota:** Se il dispositivo dell'utente dispone di una tastiera hardware collegata, il metodo di input soft non viene visualizzato.

## **Visualizzare il metodo di input su richiesta**

Se nel ciclo di vita dell'attività è presente un metodo in cui si desidera assicurarsi che il metodo di input sia visibile, è possibile utilizzare [InputMethodManager](https://developer.android.com/reference/android/view/inputmethod/InputMethodManager) per mostrarlo.

Ad esempio, il metodo seguente accetta una [View](https://developer.android.com/reference/android/view/View) in cui l'utente deve digitare qualcosa, chiama [requestFocus()](https://developer.android.com/reference/android/view/View#requestFocus()) per dargli lo stato attivo, quindi [showSoftInput()](https://developer.android.com/reference/android/view/inputmethod/InputMethodManager#showSoftInput(android.view.View,%20int)) per aprire il metodo di input:

[KOTLIN](https://developer.android.com/training/keyboard-input/visibility#kotlin)

fun showSoftKeyboard(view: View) {  
    if (view.requestFocus()) {  
        val imm = getSystemService(Context.INPUT\_METHOD\_SERVICE) as InputMethodManager  
        imm.showSoftInput(view, InputMethodManager.SHOW\_IMPLICIT)  
    }  
}

**Nota:** Una volta che il metodo di input è visibile, non dovresti nasconderlo a livello di programmazione. Il sistema nasconde il metodo di input quando l'utente termina l'attività nel campo di testo o l'utente può nasconderla con un controllo di sistema, ad esempio con il pulsante Indietro.

## **Specificare la modalità di risposta dell'interfaccia utente**

Quando il metodo di input viene visualizzato sullo schermo, riduce la quantità di spazio disponibile per l'interfaccia utente dell'app. Il sistema prende una decisione su come regolare la parte visibile dell'interfaccia utente, ma potrebbe non farlo bene. Per garantire il comportamento migliore per la tua app, devi specificare come vuoi che il sistema mostrerà l'interfaccia utente nello spazio rimanente.

Per dichiarare il trattamento preferito in un'attività, utilizzare [l'attributo android:windowSoftInputMode](https://developer.android.com/guide/topics/manifest/activity-element#wsoft)nell'elemento <activity> del manifesto con uno dei valori "adjust".

Ad esempio, per assicurarsi che il sistema ridimensiona il layout nello spazio disponibile, in modo da garantire che tutto il contenuto del layout sia accessibile (anche se probabilmente richiede lo scorrimento), utilizzare "adjustResize":

<application ... >  
    <activity  
        android:windowSoftInputMode="adjustResize" ... >  
        ...  
    </activity>  
    ...  
</application>

È possibile combinare la specifica di regolazione con la specifica [di visibilità iniziale del metodo di](https://developer.android.com/training/keyboard-input/visibility#ShowOnStart) input dall'alto:

    <activity  
        android:windowSoftInputMode="stateVisible|adjustResize" ... >  
        ...  
    </activity>

Specificare "adjustResize" è importante se l'interfaccia utente include controlli a cui l'utente potrebbe dover accedere immediatamente dopo o durante l'esecuzione dell'input di testo. Ad esempio, se usi un layout relativo per posizionare una barra dei pulsanti nella parte inferiore dello schermo, usando "adjustResize" il ridimensionamento del layout in modo che la barra dei pulsanti venga visualizzata sopra il metodo di input.

# Supportare l'esplorazione della tastiera

Oltre ai metodi di input soft (come le tastiere su schermo), Android supporta le tastiere fisiche collegate al dispositivo. Una tastiera offre non solo una comoda modalità per l'input di testo, ma offre anche un modo per gli utenti di navigare e interagire con la tua app. Sebbene la maggior parte dei dispositivi portatili come i telefoni utilizzi il tocco come modalità principale di interazione, i tablet e dispositivi simili stanno crescendo in popolarità e a molti utenti piace collegare accessori per tastiera.

Poiché più dispositivi Android offrono questo tipo di esperienza, è importante ottimizzare l'app per supportare l'interazione tramite tastiera. Questa lezione descrive come supportare meglio la navigazione con una tastiera.

**Nota:** Il supporto della navigazione direzionale nell'applicazione è anche importante per garantire che l'applicazione [sia accessibile](https://developer.android.com/guide/topics/ui/accessibility/apps) agli utenti che non navigano utilizzando segnali visivi. Il supporto completo della navigazione direzionale nell'applicazione può anche aiutarti ad automatizzare [i test dell'interfaccia](https://developer.android.com/tools/testing/testing_ui) utente con strumenti [come uiautomator](https://developer.android.com/tools/help/uiautomator).

## **Testare l'app**

È possibile che gli utenti possano già navigare nella tua app usando una tastiera, perché il sistema Android abilita la maggior parte dei comportamenti necessari per impostazione predefinita.

Tutti i widget interattivi forniti dal framework Android (ad esempio [Button](https://developer.android.com/reference/android/widget/Button) e [EditText](https://developer.android.com/reference/android/widget/EditText)) sono focalizzabili. Ciò significa che gli utenti possono navigare con dispositivi di controllo come un D-pad o una tastiera e ogni widget si illumina o cambia in altro modo il suo aspetto quando ottiene lo stato attivo per l'input.

Per testare l'app:

1. Installa la tua app in un dispositivo che offre una tastiera hardware.

Se non si dispone di un dispositivo hardware con una tastiera, collegare una tastiera Bluetooth o una tastiera USB (anche se non tutti i dispositivi supportano accessori USB).

Puoi anche usare l'emulatore Android:

* 1. In AVD Manager fare clic su **Nuovo dispositivo oppure selezionare** un profilo esistente e fare clic su **Clona**.
  2. Nella finestra visualizzata verificare che tastiera **e** **DPad siano** abilitati.

1. Per testare l'app, usa solo tab per spostarsi nell'interfaccia utente, assicurandoti che ogni controllo dell'interfaccia utente venga messa a fuoco come previsto.

Cerca tutti i casi in cui lo stato attivo si muove in un modo che non ti aspetti.

1. Inizia dall'inizio dell'app e usa invece i controlli di direzione (tasti di direzione sulla tastiera) per esplorare l'app.

Da ogni elemento attivo dell'interfaccia utente, premi Su, Giù, Sinistra e Destra.

Cerca tutti i casi in cui lo stato attivo si muove in un modo che non ti aspetti.

Se si verificano casi in cui lo spostamento con tab o controlli direzione non fa quello previsto, specificare dove deve andare lo stato attivo nel layout, come discusso nelle sezioni seguenti.

## Gestire l'esplorazione delle schede

Quando un utente naviga nell'app usando il tasto TAB della tastiera, il sistema passa lo stato attivo di input tra gli elementi in base all'ordine in cui vengono visualizzati nel layout. Se si utilizza un layout relativo, ad esempio, e l'ordine degli elementi sullo schermo è diverso dall'ordine nel file, potrebbe essere necessario specificare manualmente l'ordine di messa a fuoco.

Ad esempio, nel layout seguente, due pulsanti sono allineati sul lato destro e un campo di testo è allineato a sinistra del secondo pulsante. Per passare lo stato attivo dal primo pulsante al campo di testo, quindi al secondo pulsante, il layout deve definire in modo esplicito l'ordine di messa a fuoco per ciascuno degli elementi messa a fuoco con [l'attributo android:nextFocusForward:](https://developer.android.com/reference/android/view/View#attr_android:nextFocusForward)

<RelativeLayout ...>  
    <Button  
        android:id="@+id/button1"  
        android:layout\_alignParentTop="true"  
        android:layout\_alignParentRight="true"  
        android:nextFocusForward="@+id/editText1"  
        ... />  
    <Button  
        android:id="@+id/button2"  
        android:layout\_below="@id/button1"  
        android:nextFocusForward="@+id/button1"  
        ... />  
    <EditText  
        android:id="@id/editText1"  
        android:layout\_alignBottom="@+id/button2"  
        android:layout\_toLeftOf="@id/button2"  
        android:nextFocusForward="@+id/button2"  
        ...  />  
    ...  
</RelativeLayout>

Ora invece di inviare lo stato attivo da button1 a button2 e poi editText1, lo stato attivo si sposta in modo appropriato in base all'aspetto sullo schermo: da button1 a editText1 allora button2.

## **Gestire la navigazione direzionale**

Gli utenti possono anche esplorare l'app usando i tasti di direzione su una tastiera (il comportamento è lo stesso di quando si naviga con un D-pad o un trackball). Il sistema fornisce una migliore ipotesi su quale vista dovrebbe essere messa a fuoco in una determinata direzione in base al layout delle viste sullo schermo. A volte, tuttavia, il sistema potrebbe indovinare sbagliato.

Se il sistema non passa lo stato attivo alla vista appropriata quando si naviga in una determinata direzione, specificare quale vista deve ricevere lo stato attivo con i seguenti attributi:

* [android:nextFocusUp](https://developer.android.com/reference/android/view/View#attr_android:nextFocusUp)
* [android:nextFocusDown](https://developer.android.com/reference/android/view/View#attr_android:nextFocusDown)
* [android:nextFocusLeft](https://developer.android.com/reference/android/view/View#attr_android:nextFocusLeft)
* [android:nextFocusRight](https://developer.android.com/reference/android/view/View#attr_android:nextFocusRight)

Ogni attributo designa la visualizzazione successiva per ricevere lo stato attivo quando l'utente si sposta in quella direzione, come specificato dall'ID visualizzazione. Per esempio:

<Button  
    android:id="@+id/button1"  
    android:nextFocusRight="@+id/button2"  
    android:nextFocusDown="@+id/editText1"  
    ... />  
<Button  
    android:id="@id/button2"  
    android:nextFocusLeft="@id/button1"  
    android:nextFocusDown="@id/editText1"  
    ... />  
<EditText  
    android:id="@id/editText1"  
    android:nextFocusUp="@id/button1"  
    ...  />

# Gestire le azioni della tastiera

Quando l'utente dà il focus a una visualizzazione di testo modificabile, ad esempio un elemento [EditText](https://developer.android.com/reference/android/widget/EditText), e l'utente ha una tastiera hardware collegata, tutto l'input viene gestito dal sistema. Se, tuttavia, si desidera intercettare o gestire direttamente l'input della tastiera da soli, è possibile farlo implementando metodi di callback dall'interfaccia [KeyEvent.Callback](https://developer.android.com/reference/android/view/KeyEvent.Callback), ad esempio [onKeyDown()](https://developer.android.com/reference/android/view/KeyEvent.Callback#onKeyDown(int,%20android.view.KeyEvent)) e [onKeyMultiple()](https://developer.android.com/reference/android/view/KeyEvent.Callback#onKeyMultiple(int,%20int,%20android.view.KeyEvent)).

Entrambe le classi [Activity](https://developer.android.com/reference/android/app/Activity) e [View](https://developer.android.com/reference/android/view/View) implementano l'interfaccia [KeyEvent.Callback](https://developer.android.com/reference/android/view/KeyEvent.Callback), pertanto in genere è necessario eseguire l'override dei metodi di callback nell'estensione di queste classi in base alle esigenze.

**Nota:** Quando si gestiscono gli eventi della tastiera con la classe [**KeyEvent**](https://developer.android.com/reference/android/view/KeyEvent) e le API correlate, è necessario aspettarsi che tali eventi della tastiera provengano solo da una tastiera hardware. Non dovresti mai fare affidamento sulla ricezione di eventi chiave per qualsiasi tasto su un metodo di input soft (una tastiera su schermo).

## **Gestire eventi a chiave singola**

Per gestire un singolo tasto premere, implementare [onKeyDown()](https://developer.android.com/reference/android/app/Activity#onKeyDown(int,%20android.view.KeyEvent)) o [onKeyUp()](https://developer.android.com/reference/android/app/Activity#onKeyUp(int,%20android.view.KeyEvent)) come appropriato. Di solito, dovresti usare [onKeyUp()](https://developer.android.com/reference/android/app/Activity#onKeyUp(int,%20android.view.KeyEvent)) se vuoi essere sicuro di ricevere un solo evento. Se l'utente preme e tiene premuto il pulsante, [onKeyDown()](https://developer.android.com/reference/android/app/Activity#onKeyDown(int,%20android.view.KeyEvent)) viene chiamato più volte.

Ad esempio, questa implementazione risponde ad alcuni tasti della tastiera per controllare un gioco:

[KOTLIN](https://developer.android.com/training/keyboard-input/commands#kotlin)

override fun onKeyUp(keyCode: Int, event: KeyEvent): Boolean {  
    return when (keyCode) {  
        KeyEvent.KEYCODE\_D -> {  
            moveShip(MOVE\_LEFT)  
            true  
        }  
        KeyEvent.KEYCODE\_F -> {  
            moveShip(MOVE\_RIGHT)  
            true  
        }  
        KeyEvent.KEYCODE\_J -> {  
            fireMachineGun()  
            true  
        }  
        KeyEvent.KEYCODE\_K -> {  
            fireMissile()  
            true  
        }  
        else -> super.onKeyUp(keyCode, event)  
    }  
}

## **Tasti di modifica handle**

Per rispondere agli eventi del tasto di modifica, ad esempio quando un tasto viene combinato con MAIUSC o CTRL, è possibile eseguire una query [KeyEvent](https://developer.android.com/reference/android/view/KeyEvent) sull'oggetto passato al metodo di callback. Diversi metodi forniscono informazioni sui tasti di modifica quali [getModifiers()](https://developer.android.com/reference/android/view/KeyEvent#getModifiers()) e [getMetaState()](https://developer.android.com/reference/android/view/KeyEvent#getMetaState()). Tuttavia, la soluzione più semplice è verificare se il tasto di modifica esatto a cui si interessa viene premuto con metodi quali [isShiftPressed()](https://developer.android.com/reference/android/view/KeyEvent#isShiftPressed()) e [isCtrlPressed()](https://developer.android.com/reference/android/view/KeyEvent#isCtrlPressed()).

Ad esempio, ecco di nuovo l'implementazione di [onKeyUp()](https://developer.android.com/reference/android/app/Activity#onKeyUp(int,%20android.view.KeyEvent)), con una gestione aggiuntiva per quando il tasto MAIUSC viene tenuto premuto con una delle chiavi:

[KOTLIN](https://developer.android.com/training/keyboard-input/commands#kotlin)

override fun onKeyUp(keyCode: Int, event: KeyEvent): Boolean {  
    return when (keyCode) {  
        ...  
        KeyEvent.KEYCODE\_J -> {  
            if (event.isShiftPressed) {  
                fireLaser()  
            } else {  
                fireMachineGun()  
            }  
            true  
        }  
        KeyEvent.KEYCODE\_K -> {  
            if (event.isShiftPressed) {  
                fireSeekingMissle()  
            } else {  
                fireMissile()  
            }  
            true  
        }  
        else -> super.onKeyUp(keyCode, event)  
    }  
}

# Supportare i controller di gioco

Puoi migliorare notevolmente l'esperienza utente nel tuo gioco consentendo ai giocatori di utilizzare i loro controller di gioco preferiti. Il framework Android fornisce API per rilevare ed elaborare l'input dell'utente dai controller di gioco.

Questa classe mostra come far funzionare il tuo gioco in modo coerente con i controller di gioco su diversi livelli di API Android (livello API 9 e versioni precedenti) e come migliorare l'esperienza di gioco per i giocatori supportando più controller contemporaneamente nella tua app.

# Gestire le azioni del controller

A livello di sistema, Android segnala i codici evento di input dai controller di gioco come codici chiave Android e valori dell'asse. Nel tuo gioco, puoi ricevere questi codici e valori e convertirli in specifiche azioni di gioco.

Quando i giocatori si connettono fisicamente o accoppiano in modalità wireless un controller di gioco ai propri dispositivi basati su Android, il sistema rileva automaticamente il controller come dispositivo di input e inizia a segnalare i suoi eventi di input. Il gioco può ricevere questi eventi di input implementando i seguenti metodi di callback nell'[Activity](https://developer.android.com/reference/android/app/Activity) o nella [View](https://developer.android.com/reference/android/view/View) attiva (è necessario implementare i callback per [Activity](https://developer.android.com/reference/android/app/Activity) o [View](https://developer.android.com/reference/android/view/View), ma non per entrambi):

* Da: [Activity](https://developer.android.com/reference/android/app/Activity)
  + [dispatchGenericMotionEvent(android.view. MotionEvent)](https://developer.android.com/reference/android/app/Activity#dispatchGenericMotionEvent(android.view.MotionEvent))

Chiamato per elaborare eventi di movimento generici come i movimenti del joystick.

* + [dispatchKeyEvent(android.view.KeyEvent)](https://developer.android.com/reference/android/app/Activity#dispatchKeyEvent(android.view.KeyEvent))

Chiamato per elaborare eventi chiave come una pressione o un rilascio di un gamepad o di un pulsante D-pad.

* Da: [View](https://developer.android.com/reference/android/view/View)
  + [onGenericMotionEvent(android.view.MotionEvent)](https://developer.android.com/reference/android/view/View#onGenericMotionEvent(android.view.MotionEvent))

Chiamato per elaborare eventi di movimento generici come i movimenti del joystick.

* + [onKeyDown(int, android.view.KeyEvent)](https://developer.android.com/reference/android/view/View#onKeyDown(int,%20android.view.KeyEvent))

Chiamato per elaborare una pressione di un tasto fisico come un gamepad o un pulsante D-pad.

* + [onKeyUp(int, android.view.KeyEvent)](https://developer.android.com/reference/android/view/View#onKeyUp(int,%20android.view.KeyEvent))

Chiamato per elaborare il rilascio di una chiave fisica come un gamepad o un pulsante D-pad.

L'approccio consigliato è quello di acquisire gli eventi dalla specifica [View](https://developer.android.com/reference/android/view/View) con cui l'utente interagisce. Esaminare gli oggetti seguenti forniti dai callback per ottenere informazioni sul tipo di evento di input ricevuto:

[KeyEvent](https://developer.android.com/reference/android/view/KeyEvent)

Oggetto che descrive gli eventi del pulsante del pad direzionale (D-pad) e del gamepad. Gli eventi chiave sono accompagnati da un codice chiave che indica il pulsante specifico attivato, ad esempio [DPAD\_DOWN](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_DOWN) o [BUTTON\_A](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_A). È possibile ottenere il codice chiave chiamando [getKeyCode()](https://developer.android.com/reference/android/view/KeyEvent#getKeyCode()) o da callback di eventi chiave quali [onKeyDown()](https://developer.android.com/reference/android/view/View#onKeyDown(int,%20android.view.KeyEvent)).

[MotionEvent](https://developer.android.com/reference/android/view/MotionEvent)

Oggetto che descrive l'input dei movimenti del joystick e del grilletto della spalla. Gli eventi di movimento sono accompagnati da un codice di azione e da un insieme di valori dell'asse. Il codice azione specifica la modifica dello stato che si è verificata, ad esempio un joystick da spostare. I valori dell'asse descrivono la posizione e altre proprietà di movimento per un controllo fisico specifico, ad esempio [AXIS\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_X) o [AXIS\_RTRIGGER](https://developer.android.com/reference/android/view/MotionEvent#AXIS_RTRIGGER). È possibile ottenere il codice di azione chiamando [getAction()](https://developer.android.com/reference/android/view/MotionEvent#getAction()) e il valore dell'asse chiamando [getAxisValue()](https://developer.android.com/reference/android/view/MotionEvent#getAxisValue(int)).

Questa lezione si concentra su come gestire l'input dai tipi più comuni di controlli fisici (pulsanti del gamepad, pad direzionali e joystick) in una schermata di gioco implementando i metodi di callback della [View](https://developer.android.com/reference/android/view/View) sopra menzionati e l'elaborazione degli oggetti [KeyEvent](https://developer.android.com/reference/android/view/KeyEvent) e [MotionEvent](https://developer.android.com/reference/android/view/MotionEvent).

## Verificare che un controller di gioco sia connesso

Quando si segnalano eventi di input, Android non distingue tra eventi provenienti da un dispositivo controller non di gioco ed eventi provenienti da un controller di gioco. Ad esempio, un'azione touch screen genera un evento [AXIS\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_X) che rappresenta la coordinata X della superficie di tocco, ma un joystick genera un evento [AXIS\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_X) che rappresenta la posizione X del joystick. Se il gioco si preoccupa di gestire l'input del controller di gioco, devi prima verificare che l'evento di input provenga da un tipo di origine pertinente.

Per verificare che un dispositivo di input connesso sia un controller di gioco, chiamare [getSources()](https://developer.android.com/reference/android/view/InputDevice#getSources()) per ottenere un campo di bit combinato di tipi di origine di input supportati su tale dispositivo. È quindi possibile verificare se sono impostati i seguenti campi:

* Un tipo di origine di [SOURCE\_GAMEPAD](https://developer.android.com/reference/android/view/InputDevice#SOURCE_GAMEPAD) indica che il dispositivo di input dispone di pulsanti gamepad (ad esempio [BUTTON\_A](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_A)). Si noti che questo tipo di origine non indica rigorosamente se il controller di gioco ha pulsanti D-pad, anche se la maggior parte dei gamepad in genere ha controlli direzionali.
* Un tipo di origine di [SOURCE\_DPAD](https://developer.android.com/reference/android/view/InputDevice#SOURCE_DPAD) indica che il dispositivo di input dispone di pulsanti D-pad (ad esempio [DPAD\_UP](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_UP)).
* Un tipo di origine di [SOURCE\_JOYSTICK](https://developer.android.com/reference/android/view/InputDevice#SOURCE_JOYSTICK) indica che il dispositivo di input dispone di bastoncini di controllo analogici (ad esempio un joystick che registra i movimenti lungo [AXIS\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_X) e [AXIS\_Y](https://developer.android.com/reference/android/view/MotionEvent#AXIS_Y)).

Il frammento di codice seguente mostra un metodo di helper che consente di verificare se i dispositivi di input connessi sono controller di gioco. In tal caso, il metodo recupera gli ID dispositivo per i controller di gioco. Puoi quindi associare ogni ID dispositivo a un giocatore nel tuo gioco ed elaborare separatamente le azioni di gioco per ogni giocatore connesso. Per ulteriori informazioni sul supporto di più controller di gioco connessi contemporaneamente sullo stesso dispositivo Android, vedere [Supportare più controller di gioco](https://developer.android.com/training/game-controllers/multiple-controllers).

[KOTLIN](https://developer.android.com/training/game-controllers/controller-input#kotlin)

fun getGameControllerIds(): List<Int> {  
    val gameControllerDeviceIds = mutableListOf<Int>()  
    val deviceIds = InputDevice.getDeviceIds()  
    deviceIds.forEach { deviceId ->  
        InputDevice.getDevice(deviceId).apply {  
  
            // Verify that the device has gamepad buttons, control sticks, or both.  
            if (sources and InputDevice.SOURCE\_GAMEPAD == InputDevice.SOURCE\_GAMEPAD  
                    || sources and InputDevice.SOURCE\_JOYSTICK == InputDevice.SOURCE\_JOYSTICK) {  
                // This device is a game controller. Store its device ID.  
                gameControllerDeviceIds  
                        .takeIf { !it.contains(deviceId) }  
                        ?.add(deviceId)  
            }  
        }  
    }  
    return gameControllerDeviceIds  
}

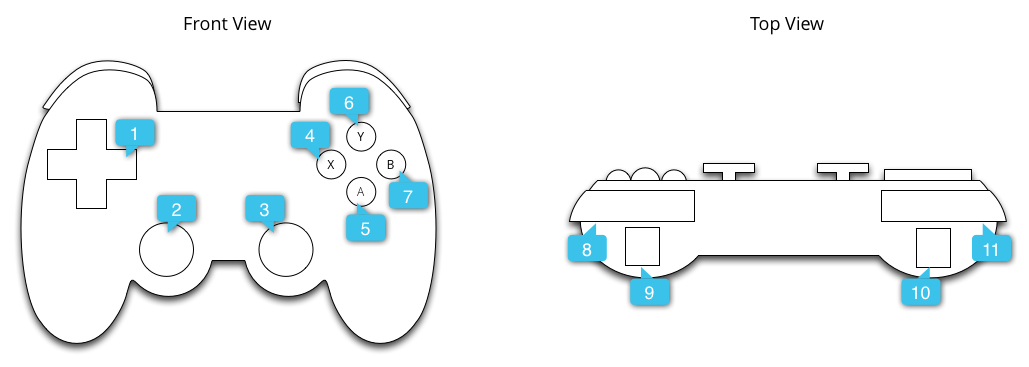
Inoltre, potresti voler verificare la presenza di singole funzionalità di input supportate da un controller di gioco connesso. Questo potrebbe essere utile, ad esempio, se vuoi che il tuo gioco usi solo l'input dal set di controlli fisici che comprende.

Per rilevare se un codice chiave o un codice asse specifico è supportato da un controller di gioco connesso, utilizzare queste tecniche:

* In Android 4.4 (livello API 19) o superiore, è possibile determinare se un codice chiave è supportato su un controller di gioco connesso chiamando [hasKeys(int...)](https://developer.android.com/reference/android/view/InputDevice#hasKeys(int...)) .
* In Android 3.1 (livello API 12) o superiore, puoi trovare tutti gli assi disponibili supportati su un controller di gioco connesso chiamando prima [getMotionRanges()](https://developer.android.com/reference/android/view/InputDevice#getMotionRanges()). Quindi, su ogni oggetto [InputDevice.MotionRange](https://developer.android.com/reference/android/view/InputDevice.MotionRange) restituito, chiamare [getAxis()](https://developer.android.com/reference/android/view/InputDevice.MotionRange#getAxis()) per ottenere l'ID asse.

## **Pulsante Elabora gamepad preme**

La figura 1 mostra come Android mappa i codici chiave e i valori dell'asse ai controlli fisici sulla maggior parte dei controller di gioco.



**Figura 1.** Profilo per un controller di gioco generico.

I callout nella figura fanno riferimento a quanto segue:

1. [AXIS\_HAT\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_HAT_X), , , , ,[AXIS\_HAT\_Y](https://developer.android.com/reference/android/view/MotionEvent#AXIS_HAT_Y)[DPAD\_UP](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_UP)[DPAD\_DOWN](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_DOWN)[DPAD\_LEFT](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_LEFT)[DPAD\_RIGHT](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_RIGHT)
2. [AXIS\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_X), , [AXIS\_Y](https://developer.android.com/reference/android/view/MotionEvent#AXIS_Y)[BUTTON\_THUMBL](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_THUMBL)
3. [AXIS\_Z](https://developer.android.com/reference/android/view/MotionEvent#AXIS_Z), , [AXIS\_RZ](https://developer.android.com/reference/android/view/MotionEvent#AXIS_RZ)[BUTTON\_THUMBR](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_THUMBR)
4. [BUTTON\_X](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_X)
5. [BUTTON\_A](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_A)
6. [BUTTON\_Y](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_Y)
7. [BUTTON\_B](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_B)
8. [BUTTON\_R1](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_R1)
9. [AXIS\_RTRIGGER](https://developer.android.com/reference/android/view/MotionEvent#AXIS_RTRIGGER), [AXIS\_THROTTLE](https://developer.android.com/reference/android/view/MotionEvent#AXIS_THROTTLE)
10. [AXIS\_LTRIGGER](https://developer.android.com/reference/android/view/MotionEvent#AXIS_LTRIGGER), [AXIS\_BRAKE](https://developer.android.com/reference/android/view/MotionEvent#AXIS_BRAKE)
11. [BUTTON\_L1](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_L1)

I codici tasto comuni generati dalle pressioni dei pulsanti del gamepad includono [BUTTON\_A](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_A), [BUTTON\_B](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_B), [BUTTON\_SELECT](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_SELECT) e [BUTTON\_START](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_START). Alcuni controller di gioco attivano anche il codice tasto [DPAD\_CENTER](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_CENTER) quando viene premuto il centro della traversa del D-pad. Il gioco può esaminare il codice chiave chiamando [getKeyCode()](https://developer.android.com/reference/android/view/KeyEvent#getKeyCode()) o da callback di eventi chiave come e [onKeyDown()](https://developer.android.com/reference/android/view/View#onKeyDown(int,%20android.view.KeyEvent)), se rappresenta un evento rilevante per il gioco, elaborarlo come azione di gioco. La tabella 1 elenca le azioni di gioco consigliate per i pulsanti più comuni del gamepad.

**La tabella 1.** Azioni di gioco consigliate per i pulsanti del gamepad.

|  |  |
| --- | --- |
| **Azione di gioco** | **Codice tasto pulsante** |
| Inizia il gioco nel menu principale o metti in pausa /annulla la pausa durante il gioco | [BUTTON\_START](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_START)\* |
| Menu Visualizza | [BUTTON\_SELECT](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_SELECT)\* e [KEYCODE\_MENU](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_MENU)\* |
| Come il comportamento di navigazione indietro di Android descritto nella guida [alla progettazione](https://developer.android.com/design/patterns/navigation) della navigazione. | [KEYCODE\_BACK](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BACK) |
| Tornare a una voce precedente in un menu | [BUTTON\_B](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_B) |
| Confermare la selezione o eseguire l'azione di gioco principale | [BUTTON\_A](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_A) e [DPAD\_CENTER](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_CENTER) |

\* Il tuo gioco non deve fare affidamento sulla presenza dei pulsanti Start, Select o Menu.

**Suggerimento:** Prendi in considerazione la possibilità di fornire una schermata di configurazione nel tuo gioco per consentire agli utenti di personalizzare i propri mapping dei controller di gioco per le azioni di gioco.

Il frammento seguente mostra come è possibile eseguire l'override [onKeyDown()](https://developer.android.com/reference/android/view/View#onKeyDown(int,%20android.view.KeyEvent)) per associare le pressioni dei pulsanti [BUTTON\_A](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_A) e [DPAD\_CENTER](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_CENTER) a un'azione di gioco.

[KOTLIN](https://developer.android.com/training/game-controllers/controller-input#kotlin)

class GameView(...) : View(...) {  
    ...  
  
    override fun onKeyDown(keyCode: Int, event: KeyEvent): Boolean {  
        var handled = false  
        if (event.source and InputDevice.SOURCE\_GAMEPAD == InputDevice.SOURCE\_GAMEPAD) {  
            if (event.repeatCount == 0) {  
                when (keyCode) {  
                    // Handle gamepad and D-pad button presses to navigate the ship  
                    ...  
  
                    else -> {  
                        keyCode.takeIf { isFireKey(it) }?.run {  
                            // Update the ship object to fire lasers  
                            ...  
                            handled = true  
                        }  
                    }  
                }  
            }  
            if (handled) {  
                return true  
            }  
        }  
        return super.onKeyDown(keyCode, event)  
    }  
  
    // Here we treat Button\_A and DPAD\_CENTER as the primary action  
    // keys for the game.  
    private fun isFireKey(keyCode: Int): Boolean =  
            keyCode == KeyEvent.KEYCODE\_DPAD\_CENTER || keyCode == KeyEvent.KEYCODE\_BUTTON\_A  
}

**Nota:** Su Android 4.2 (livello API 17) e versioni inferiori, il sistema tratta [**BUTTON\_A**](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_A) come la chiave Posteriore Android per impostazione predefinita. Se la tua app supporta queste versioni di Android, assicurati di trattare [**BUTTON\_A**](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_BUTTON_A) come l'azione di gioco principale. Per determinare la versione corrente di Android SDK nel dispositivo, fare riferimento al valore [**Build.VERSION.SDK\_INT**](https://developer.android.com/reference/android/os/Build.VERSION#SDK_INT).

## **Ingresso del pad direzionale di processo**

Il pad direzionale a 4 direzioni (D-pad) è un controllo fisico comune in molti controller di gioco. Android segnala che D-pad UP e DOWN preme come eventi [AXIS\_HAT\_Y](https://developer.android.com/reference/android/view/MotionEvent#AXIS_HAT_Y) con un intervallo da -1.0 (fino a) a 1.0 (in basso) e D-pad LEFT o RIGHT preme come eventi [AXIS\_HAT\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_HAT_X) con un intervallo da -1.0 (a sinistra) a 1.0 (a destra).

Alcuni controller segnalano invece le pressioni del D-pad con un codice tasto. Se il gioco si preoccupa delle presse D-pad, è necessario trattare gli eventi dell'asse del cappello e i codici tasto D-pad come gli stessi eventi di input, come raccomandato nella tabella 2.

**La tabella 2.** Azioni di gioco predefinite consigliate per i codici tasto D-pad e i valori dell'asse del cappello.

|  |  |  |
| --- | --- | --- |
| **Azione di gioco** | **Codice tasto D-pad** | **Codice asse hat** |
| promuovere | [KEYCODE\_DPAD\_UP](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_UP) | [AXIS\_HAT\_Y](https://developer.android.com/reference/android/view/MotionEvent#AXIS_HAT_Y) (per i valori da 0 a -1,0) |
| retrocedere | [KEYCODE\_DPAD\_DOWN](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_DOWN) | [AXIS\_HAT\_Y](https://developer.android.com/reference/android/view/MotionEvent#AXIS_HAT_Y) (per i valori da 0 a 1,0) |
| Spostarsi a sinistra | [KEYCODE\_DPAD\_LEFT](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_LEFT) | [AXIS\_HAT\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_HAT_X) (per i valori da 0 a -1,0) |
| Spostarsi a destra | [KEYCODE\_DPAD\_RIGHT](https://developer.android.com/reference/android/view/KeyEvent#KEYCODE_DPAD_RIGHT) | [AXIS\_HAT\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_HAT_X) (per i valori da 0 a 1,0) |

Il frammento di codice seguente mostra una classe helper che consente di controllare l'asse del cappello e i valori del codice chiave da un evento di input per determinare la direzione del D-pad.

[KOTLIN](https://developer.android.com/training/game-controllers/controller-input#kotlin)

class Dpad {  
  
    private var directionPressed = -1 // initialized to -1  
  
    fun getDirectionPressed(event: InputEvent): Int {  
        if (!isDpadDevice(event)) {  
            return -1  
        }  
  
        // If the input event is a MotionEvent, check its hat axis values.  
        (event as? MotionEvent)?.apply {  
  
            // Use the hat axis value to find the D-pad direction  
            val xaxis: Float = event.getAxisValue(MotionEvent.AXIS\_HAT\_X)  
            val yaxis: Float = event.getAxisValue(MotionEvent.AXIS\_HAT\_Y)  
  
            directionPressed = when {  
                // Check if the AXIS\_HAT\_X value is -1 or 1, and set the D-pad  
                // LEFT and RIGHT direction accordingly.  
                xaxis.compareTo(-1.0f) == 0 -> Dpad.LEFT  
                xaxis.compareTo(1.0f) == 0 -> Dpad.RIGHT  
                // Check if the AXIS\_HAT\_Y value is -1 or 1, and set the D-pad  
                // UP and DOWN direction accordingly.  
                yaxis.compareTo(-1.0f) == 0 -> Dpad.UP  
                yaxis.compareTo(1.0f) == 0 -> Dpad.DOWN  
                else -> directionPressed  
            }  
        }  
        // If the input event is a KeyEvent, check its key code.  
        (event as? KeyEvent)?.apply {  
  
            // Use the key code to find the D-pad direction.  
            directionPressed = when(event.keyCode) {  
                KeyEvent.KEYCODE\_DPAD\_LEFT -> Dpad.LEFT  
                KeyEvent.KEYCODE\_DPAD\_RIGHT -> Dpad.RIGHT  
                KeyEvent.KEYCODE\_DPAD\_UP -> Dpad.UP  
                KeyEvent.KEYCODE\_DPAD\_DOWN -> Dpad.DOWN  
                KeyEvent.KEYCODE\_DPAD\_CENTER ->  Dpad.CENTER  
                else -> directionPressed  
            }  
        }  
        return directionPressed  
    }  
  
    companion object {  
        internal const val UP = 0  
        internal const val LEFT = 1  
        internal const val RIGHT = 2  
        internal const val DOWN = 3  
        internal const val CENTER = 4  
  
        fun isDpadDevice(event: InputEvent): Boolean =  
            // Check that input comes from a device with directional pads.  
            event.source and InputDevice.SOURCE\_DPAD != InputDevice.SOURCE\_DPAD  
    }  
}

Puoi usare questa classe di supporto nel tuo gioco ovunque tu voglia elaborare l'input del D-pad (ad esempio, nei callback [onGenericMotionEvent()](https://developer.android.com/reference/android/view/View#onGenericMotionEvent(android.view.MotionEvent)) o [onKeyDown()](https://developer.android.com/reference/android/view/View#onKeyDown(int,%20android.view.KeyEvent))).

Per esempio:

[KOTLIN](https://developer.android.com/training/game-controllers/controller-input#kotlin)

private val dpad = Dpad()  
...  
override fun onGenericMotionEvent(event: MotionEvent): Boolean {  
    if (Dpad.isDpadDevice(event)) {  
        when (dpad.getDirectionPressed(event)) {  
            Dpad.LEFT -> {  
                // Do something for LEFT direction press  
                ...  
                return true  
            }  
            Dpad.RIGHT -> {  
                // Do something for RIGHT direction press  
                ...  
                return true  
            }  
            Dpad.UP -> {  
                // Do something for UP direction press  
                ...  
                return true  
            }  
            ...  
        }  
    }  
  
    // Check if this event is from a joystick movement and process accordingly.  
    ...  
}

## **Elaborare i movimenti del joystick**

Quando i giocatori spostano un joystick sui loro controller di gioco, Android segnala un [MotionEvent](https://developer.android.com/reference/android/view/MotionEvent) che contiene il codice di azione [ACTION\_MOVE](https://developer.android.com/reference/android/view/MotionEvent#ACTION_MOVE) e le posizioni aggiornate degli assi del joystick. Il tuo gioco può utilizzare i dati forniti da [MotionEvent](https://developer.android.com/reference/android/view/MotionEvent) per determinare se un movimento del joystick a cui importa è accaduto.

Si noti che gli eventi di movimento del joystick possono mettere insieme più campioni di movimento all'interno di un singolo oggetto. L'oggetto [MotionEvent](https://developer.android.com/reference/android/view/MotionEvent) contiene la posizione corrente per ogni asse joystick e più posizioni cronologiche per ogni asse. Quando si segnalano eventi di movimento con codice di azione [ACTION\_MOVE](https://developer.android.com/reference/android/view/MotionEvent#ACTION_MOVE) (ad esempio movimenti del joystick), Android fa in batch i valori dell'asse per l'efficienza. I valori cronologici per un asse sono costituiti dall'insieme di valori distinti precedenti al valore dell'asse corrente e più recenti dei valori riportati in qualsiasi evento di movimento precedente. Vedi il riferimento [MotionEvent](https://developer.android.com/reference/android/view/MotionEvent) per i dettagli.

Puoi usare le informazioni storiche per eseguire il rendering più accurato del movimento di un oggetto di gioco in base all'input del joystick. Per recuperare i valori correnti e cronologici, chiamare [getAxisValue()](https://developer.android.com/reference/android/view/MotionEvent#getAxisValue(int)) o [getHistoricalAxisValue()](https://developer.android.com/reference/android/view/MotionEvent#getHistoricalAxisValue(int,%20int)). È inoltre possibile trovare il numero di punti storici nell'evento joystick chiamando [getHistorySize()](https://developer.android.com/reference/android/view/MotionEvent#getHistorySize()).

Nel frammento seguente viene illustrato come eseguire l'override del callback [onGenericMotionEvent()](https://developer.android.com/reference/android/view/View#onGenericMotionEvent(android.view.MotionEvent)) per elaborare l'input del joystick. È innanzitutto necessario elaborare i valori cronologici per un asse, quindi elaborarne la posizione corrente.

[KOTLIN](https://developer.android.com/training/game-controllers/controller-input#kotlin)

class GameView(...) : View(...) {  
  
    override fun onGenericMotionEvent(event: MotionEvent): Boolean {  
  
        // Check that the event came from a game controller  
        return if (event.source and InputDevice.SOURCE\_JOYSTICK == InputDevice.SOURCE\_JOYSTICK  
                && event.action == MotionEvent.ACTION\_MOVE) {  
  
            // Process the movements starting from the  
            // earliest historical position in the batch  
            (0 until event.historySize).forEach { i ->  
                // Process the event at historical position i  
                processJoystickInput(event, i)  
            }  
  
            // Process the current movement sample in the batch (position -1)  
            processJoystickInput(event, -1)  
            true  
        } else {  
            super.onGenericMotionEvent(event)  
        }  
    }  
}

Prima di utilizzare l'input del joystick, è necessario determinare se il joystick è centrato, quindi calcolare i movimenti dell'asse di conseguenza. I joystick hanno tipicamente un'area piatta, cioè un intervallo di valori vicino alla coordinata (0,0) in corrispondenza della quale l'asse è considerato centrato. Se il valore dell'asse riportato da Android rientra nell'area piatta, è necessario trattare il controller come a riposo (cioè immobile lungo entrambi gli assi).

Lo snippet riportato di seguito mostra un metodo di aiuto che calcola il movimento lungo ogni asse. Questo helper viene vocato nel metodo processJoystickInput() descritto di seguito.

[KOTLIN](https://developer.android.com/training/game-controllers/controller-input#kotlin)

private fun getCenteredAxis(  
        event: MotionEvent,  
        device: InputDevice,  
        axis: Int,  
        historyPos: Int  
): Float {  
    val range: InputDevice.MotionRange? = device.getMotionRange(axis, event.source)  
  
    // A joystick at rest does not always report an absolute position of  
    // (0,0). Use the getFlat() method to determine the range of values  
    // bounding the joystick axis center.  
    range?.apply {  
        val value: Float = if (historyPos < 0) {  
            event.getAxisValue(axis)  
        } else {  
            event.getHistoricalAxisValue(axis, historyPos)  
        }  
  
        // Ignore axis values that are within the 'flat' region of the  
        // joystick axis center.  
        if (Math.abs(value) > flat) {  
            return value  
        }  
    }  
    return 0f  
}

Mettendo tutto insieme, ecco come potresti elaborare i movimenti del joystick nel tuo gioco:

[KOTLIN](https://developer.android.com/training/game-controllers/controller-input#kotlin)

private fun processJoystickInput(event: MotionEvent, historyPos: Int) {  
  
    val inputDevice = event.device  
  
    // Calculate the horizontal distance to move by  
    // using the input value from one of these physical controls:  
    // the left control stick, hat axis, or the right control stick.  
    var x: Float = getCenteredAxis(event, inputDevice, MotionEvent.AXIS\_X, historyPos)  
    if (x == 0f) {  
        x = getCenteredAxis(event, inputDevice, MotionEvent.AXIS\_HAT\_X, historyPos)  
    }  
    if (x == 0f) {  
        x = getCenteredAxis(event, inputDevice, MotionEvent.AXIS\_Z, historyPos)  
    }  
  
    // Calculate the vertical distance to move by  
    // using the input value from one of these physical controls:  
    // the left control stick, hat switch, or the right control stick.  
    var y: Float = getCenteredAxis(event, inputDevice, MotionEvent.AXIS\_Y, historyPos)  
    if (y == 0f) {  
        y = getCenteredAxis(event, inputDevice, MotionEvent.AXIS\_HAT\_Y, historyPos)  
    }  
    if (y == 0f) {  
        y = getCenteredAxis(event, inputDevice, MotionEvent.AXIS\_RZ, historyPos)  
    }  
  
    // Update the ship object based on the new x and y values  
}

Per supportare i controller di gioco che hanno funzionalità più sofisticate oltre a un singolo joystick, segui queste procedure consigliate:

* **Maneggiare bastoncini a doppio controller.** Molti controller di gioco hanno un joystick sinistro e destro. Per il bastone sinistro, Android segnala i movimenti orizzontali come eventi [AXIS\_X](https://developer.android.com/reference/android/view/MotionEvent#AXIS_X) e movimenti verticali come eventi [AXIS\_Y](https://developer.android.com/reference/android/view/MotionEvent#AXIS_Y). Per il bastone destro, Android segnala i movimenti orizzontali come eventi [AXIS\_Z](https://developer.android.com/reference/android/view/MotionEvent#AXIS_Z) e movimenti verticali come eventi [AXIS\_RZ](https://developer.android.com/reference/android/view/MotionEvent#AXIS_RZ). Assicurati di gestire entrambi i bastoncini del controller nel codice.
* **Gestire le presse del grilletto a spalla (ma fornire metodi di input alternativi).** Alcuni controller hanno grilletti della spalla sinistra e destra. Se questi trigger sono presenti, Android segnala una pressione del trigger sinistro come evento [AXIS\_LTRIGGER](https://developer.android.com/reference/android/view/MotionEvent#AXIS_LTRIGGER) e una pressione del trigger destro come evento [AXIS\_RTRIGGER](https://developer.android.com/reference/android/view/MotionEvent#AXIS_RTRIGGER). In Android 4.3 (API level 18), un controller che produce un [AXIS\_LTRIGGER](https://developer.android.com/reference/android/view/MotionEvent#AXIS_LTRIGGER) segnala anche un valore identico per l'asse [AXIS\_BRAKE](https://developer.android.com/reference/android/view/MotionEvent#AXIS_BRAKE). Lo stesso vale per [AXIS\_RTRIGGER](https://developer.android.com/reference/android/view/MotionEvent#AXIS_RTRIGGER) e [AXIS\_GAS](https://developer.android.com/reference/android/view/MotionEvent#AXIS_GAS). Android segnala tutte le presse analogiche con un valore normalizzato da 0,0 (rilasciato) a 1,0 (completamente premuto). Non tutti i controller hanno trigger, quindi considera di consentire ai giocatori di eseguire quelle azioni di gioco con altri pulsanti.