IJCSI International Journal of Computer Science Issues, Volume 14, Issue 3, May 2017
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.20943/01201703.113

Localization to Bidirectional Languages for a Visual Programming Environment on Smartphones

Aiman M. Ayyal Awwad

Institute of Software Technology, Graz University of Technology, 8010 Graz, Austria

Abstract

Catrobat is a free and open source visual programming environment for smartphones. It has been developed for educational purposes to help students visualize and understand learning material. It allows students to build their own animations and games for their classes in academic subjects and wirelessly control external hardware. Catrobat needs to talk to the young children in schools in their native language and enable them to get the best experience in the language of their choice. In this paper, we localize Catrobat into bidirectional languages such as Arabic, Persian, Urdu, and Hebrew, and introduce the challenging aspects of localization to such languages. The localization testing results show that the product is cosmetically correct, linguistically accurate, and culturally appropriate. Therefore, it meets bidirectional requirements, complies with bidirectionality design guidelines, and can be employed in programming education for young schoolchildren.

Keywords: *Localization*, *Bidirectional Languages*, *Visual Programming Environment*, *Smartphone*, *Educational*.

1. Introduction

To design effective educational apps, solve basic problems, and attract inexpert developers to build interactive applications a visual programming environment can be used [1]. Visual programming environments (VPEs) let users create programs by manipulating program elements graphically instead of specifying them textually [1, 2].

In fact, VPEs are very popular in programming education for young children like primary and secondary schools, since they do not require previous knowledge of programming syntax and create an environment where compile-time errors are nonexistent [2]. It is important to investigate the characteristics of VPEs as they used in the programming education for young children. When using VPEs, it is relatively easy for the young children to create simple games since there are usually built-in libraries in the language environments [2, 3].

In recent years, there are many so-called VPEs, which have been developed for educational objectives. Examples of those VPEs include Scratch, Kodu, Snap!, and Catrobat [2, 3]. They are not only easy to learn, but also introduce rich and charming visual outcomes.

Pocket Code, Catrobat's version for Android platform, is a learning application for mobile devices that has been developed in Austria at Graz University of Technology [3]. With Pocket Code children and young people can create their own games, stories, animations, interactive music videos, and many types of other apps, directly on their phones or tablets, and taking advantage of multi-touch, built-in sensors, and the full display resolution of the device. It uses a visual programming language and it is developed by the free and open source project. Also, it is featured by Google on Google Play for education platform [4, 5].

Catrobat lets students to program and design games for their classes in academic subjects such as science, math, and languages by effectively developing and adapting the learning material themselves. On various occasions, children and teenagers were being taught to program their own applications [3, 4]. They had a lot of fun using VPE to build a simple game without any previous knowledge about programming and thus, not just being users but being developers too [3, 6].

Nowadays, many countries in the world are promoting programming education for primary and secondary schoolchildren [7]. Some countries have already adopted programming as a major subject in primary education. In those countries, almost always VPEs are employed to teach programming to the children [2, 7]. Therefore, the release of VPEs on a worldwide scale requires them to be made available in many languages, including bidirectional languages such as Arabic, Persian, Urdu, and Hebrew. This asks for internationalization (I18n) and localization (L10n) of the product.

In software development, internationalization and localization are processes of adapting computer software for non-native regions, especially other locales, cultures and environments [8].

Localization of software for bidirectional languages has still not reached its full potential due to a shortage of research [8]. At present, most of the localization of VPEs is only expressed in language translation and adaptation of time, date, number, and currency formats. There are many challenging aspects of the bidirectional languages which have been neglected during localization. Until recently, many of these aspects were considered non-essential. However, with the wide-spread of the smartphones and the increased use of mobile applications in business, education, gaming, communication, and engaging in social networking among bidirectional languages speaking populations, many of these aspects are becoming necessary, even expected requirements.

The localized VPE should meet a local user's expectations in terms of language, culture, and user experience. However, Catrobat has specific issues that should be considered during the localization process. To enhance the usability of localized smartphone's applications, Google develops the Material Design guidelines for bidirectionality. In these guidelines, UI layout for languages that are read and written from right-to-left, such as Arabic and Urdu, should be mirrored to ensure content is easy to understand [9].

In this paper, the visual programming environment (Catrobat) is localized into bidirectional in general and specifically to the Arabic language (Arabization), furthermore all the challenging aspects of the bidirectional, which facing the application's developers, are introduced as well as the comprehensive solutions to these challenges are proposed. Without considering these challenges by the software localization specialist, the quality of bidirectional languages product will continue to lag behind other languages. The bidirectional languages localization testing results show that the product is cosmetically correct, linguistically accurate, and culturally appropriate. Therefore, it meets bidirectional requirements, complies with bidirectionality design guidelines, and can be employed in programming education for young schoolchildren.

2. Internationalization and Localization

Nowadays, software applications always need to be developed and deployed to different regions of the world. To be successful in domination the global market, software vendors need to develop world-ready products. However, the local version of application help local customers better understand and use it, attract more customers and maximize application sales [10].

Software internationalization and localization are important steps in deploying software to different locales of the world. Internationalization refers to the process of re-engineering a system that supports different languages and regions. Localization refers to the process of adapting an internationalized software for use in a specific country, region, or culture, by adding local-specific features and translated text [8, 10].

However, localization is not just about translating all strings and customizing of the software user interface (UI) but also making sure the product matches to a local user experience. It aimed at developing an accessible, usable and culturally suitable product for a specific locale [8]. According to data collected by Google and Admob in March 2014, it was found that 48 percent of app users in the United States had stopped using an app because of insufficient localization [11].

The difference between internationalization and localization is subtle but substantial. Internationalization is the adaptation of products to be used everywhere, while localization is the process of translating an application's resources into localized products for each locale that the application will support and adding special features for use in a specific locale [12]. Practically, internationalization process is performed once per product. Whereas localization process is performed once for every combination of product and locale. The two processes are complementary and must be integrated to lead the purpose of a system that works globally [8, 13].

Delaying localization process to the end of the software development cycle will very likely delay the shipping date because of unforeseen localization defects that require code changes and UI customization [8]. Furthermore, one localized version of the software can be used in more than one country, as long as these countries have the same language. For example, the French language is used in more than 39 countries, and the Arabic language is used in more than 23 countries [14]. Internationalization is very cost effective by having key tasks performed just once, which decreases the effort and time to localize an application into other languages [8, 12].

In designing phase it is important to consider features and peculiarities of different languages. Some of the issues that the internationalization needs to consider include [8, 10]:

- Locale and culture awareness (formats for numbers, dates, times, addresses, and units of measurement)
- Layout direction (left-to-right script vs. right-to-left)
- Complex scripts awareness
- Language character coding sets
- Sorting and indexing rules
- Case conversion

The better the application is internationalized, the easier it is localized [8, 10]. This is because an application that internationalized will have built-in support for elements which are needed for localization. These may include [8]

- Language translation
- Aesthetics (layout direction)
- Visual language (logos and icons that communicate direction)
- Spelling variants (localization (en-US, en-CA) vs. localization (en-GB, en-AU)
- Locales (e.g. displays of money amounts. In the U.S. (1,234.56). In Germany (1.234,56)
- Cultural values and social context

The colors are also another issue to consider in the internationalization of the application. Sometimes, colors express cultural meanings that need to be analyzed in apps localization. The same color may have different meanings in different cultures. For instance, red color means danger in U.S, but in China it means happiness. Similarly, in U.S. while yellow means cowardice, in Egypt it means prosperity [13].

Often applications are distributed to many countries and regions. Every country or region has its own language and culture; accordingly, they should be localized to meet all demands of local users. So what is behind the cloud in Fig. 1 is the localization process that guarantees their usability and robustness.

There are three major steps to localize application successfully [8]:

1. Translating all user-facing text strings in the application to specific language before it is displayed to the user.
2. Customizing graphics, icons, colors, fonts, styles, menus and if necessary the text rendering of each user interface element for different cultures to display text correctly and matches the intended specifications and expectations of local users.
3. Adding a locale that guarantees that all country-specific formats are shown correctly.

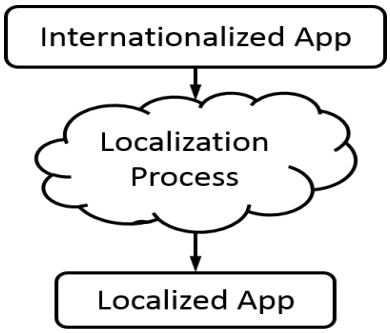


Fig. 1 The localization process.

3. Characteristics of the Arabic Writing System

This section covers the major characteristics of the Arabic and other bidirectional languages. In the area of app localization, the Arabic language still remains one of the most challenging languages [12, 15]. Although the Arabic language is right-to-left (RTL), it is also bidirectional (BiDi), which means that numbers and text in Latin based characters will display left-to-right (LTR). The Arabic language is cursive, bidirectional, and context dependent [8, 12].

3.1 Bidirectionality and Characters Reordering

Arabic scripts (used for languages such as Arabic, Persian, Pashto, Urdu, and other) and Hebrew scripts (used for Hebrew, Yiddish, and others) are read from right-to-left, further, numbers are read from left to right with the highest order digit on the left side. This leads to a mixed direction of text parts. However, it is commonly expected by readers of such languages for the scripts to be right-aligned [8, 15].

For these particular scripts, the logical order (the order in which the user enters text with a sequence of virtual-key inputs) and the visual order (the order in which characters are rendered on the screen) are different in most cases (see Fig. 2). Character positioning and a hat character movement in bidirectional context, in which RTL characters and LTR characters coexist, are the significant challenges to solving when dealing with RTL scripts. A bidirectional context can be a mixture of Latin and Arabic or Hebrew text, or it can include Arabic and Hebrew characters with numbers that have an LTR characteristic in Arabic and Hebrew [8].

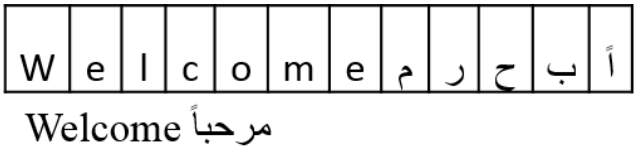


Fig. 2 Arabic text (bidirectional) where the logical order in the first row and the visual order in the second row are not of the same sequence of characters.

3.2 Contextual Shaping

The Arabic character font is cursive which means that the letters are connected together like English handwriting. For the Arabic and Indic families of languages, a character's glyph (that is, all the character's different possible representations) can form in four different shapes depending on the glyph's position in a word and the surrounding characters. Whether the letters are in initial, medial, end or isolated form, they will take on different

shapes [16, 17]. An example of the four different shapes of the Arabic character "ض" (Daad/ d) is illustrated in Table 1:

Table 1: Four different shapes of the Arabic character "ض"

Initial Form	*Middle Form*	*End Form*	*Isolated Form*	*Unicode*
ضـ	ـضـ	ـض	ض	0636

The challenging part of contextual shaping is that, for all the different glyphs, there is only one defined code point in different encoding standards. Layout and rendering mechanisms should determine (at run time) the proper glyph to be used from the font tables, depending on the context.

3.3 Ligatures

For Latin script, there is a direct one-to-one mapping between a character and its glyph. (For example, the character "s" is always rendered by the same glyph "s"). For complex scripts, groups of two or sometimes three characters linked together to form a new glyph independent of the original characters [15, 17]. An example of the "lam-alif" ligature is illustrated in Table 2:

Table 2: "lam-alif" ligature

Characters in Arabic	*Contextual Unicode values*	*Isolated Form*
ل + ا	0644+0627	لا

4. Challenges of Localizing a Mobile App

The most important issues in localizing software to bidirectional languages can be solved during the design phase of the mobile app itself. Mobile apps should ideally be designed with features that will authorize them to be adapted into as many locales as possible without great engineering changes. Here are the main challenges encountered in localizing apps into bidirectional languages.

Character encoding: The mobile apps should be able to display bidirectional languages characters as well as accept languages input from users. It is highly recommended to use Unicode scheme.

Right-to-left and vertical text: Right-to-left text for languages like Arabic, Persian, Urdu and Hebrew, and vertical text for Asian languages, need specific UI layout. The right-to-left script will also require mirroring of direction-sensitive graphics, while vertical script consumes more screen space. In addition, mixed languages' strings (for example, as a result of brand names or hyperlinks, which usually remain in English) or bidirectionality require extensive extra adaptation to support these languages.

Mobile phone screen size: One of the most important challenges presented by mobile phones is the limited screen size. Due to this issue, mobile apps' text should be very minimal and developers, UI designers, and translators should take into consideration the short user interface strings.

Font style for mobile applications: Localized product might have various UI fonts' types than the original products. Local users will use the best font type according to the font capabilities of the particular target language. Font styles for text are discouraged in layout design such as special fonts, italic, or bold, because they may affect the readability of complex characters in some languages.

Sometimes, specific fonts are too small and may require being increased in size for reading-clearness. On the other hand, some fonts are too big and may require being decreased. There can also be a critical issue with special characters, for example, umlauts, cedillas and other characters, not rendering correctly.

Text expansion: The software designer should leave enough space to allow for text expansion. Translation into Arabic and other bidirectional languages cause approximately a 30% text expansion [8]. A layout flexibility of most dialog boxes and screens is necessary. One of the reasons for the text expansion is that when the source strings containing abbreviations which do not have parallel abbreviations in the target languages, for instance in English language M/F abbreviations stand for Male/Female and there are no parallel abbreviations in the Arabic language for Male/Female, the same for the abbreviations such as (GUI, DPI, GB).

Regional standards: Mobile applications should support bidirectional languages locale standards as dates, times, currency, addresses, phone numbers and addresses formats and calendar settings, as well as sorting and indexing rules.

Search and replace: There are special search rules that are specific to bidirectional languages (for example, Hamza, a character in Arabic orthography used to represent the sound of a glottal stop, transliterated in English as an apostrophe).

Icons, symbols, and images: Although the use of graphics and icons instead of text can create a more international look and feel, it can also create some problems in localization. For instance, icons represent fingers such as an OK sign or V-sign may mean different things to different cultures. Indeed not all icons, symbols or images have a broad meaning. Special consideration must be given to the

meaning of each icon/symbol/image and even color in the target culture and make sure they are culturally appropriate.

5. Visual Programming Environment for Children: Catrobat

Catrobat is a set of creativity tools for smartphones, tablets, and mobile browsers. Through Catrobat, computational thinking skills, as well as the free and open source software philosophy, are promoted in a fun and engaging way on a worldwide scale. Catrobat and the software developed by the Catrobat team are inspired by, but distinct from, the Scratch programming system developed by the Lifelong Kindergarten Group at the MIT Media Lab. Catrobat itself is an independent free and open source software project [3, 5].

Catrobat provides an easy possibility for the children and teenagers to learn to program without any programming knowledge, and enables them to program and wirelessly control Lego EV3, Phiro, and Arduino via Bluetooth [18]. In fact, Catrobat is created for children. Since it is created for children, it is very easy to learn and use. They can create animations. For older children or teenagers, they can create Catrobat games with single-level or multi-levels. In addition, teachers and students in schools can use Catrobat to design effective education programs such as math quiz, physics simulation, and educational videos [5, 6].

The version of Catrobat which is developed for Android smartphones is named Catroid and is available on Google's Play Store as "Pocket Code". Pocket Code allows children to create and execute Catrobat programs on Android, iOS, and Windows Phone 8 smartphones as well as on HTML5 capable mobile browsers. Catrobat's programs written on one platform can be directly run on all other platforms as well and can be shared via the Pocket Code sharing website. The Android version of Pocket Code currently works well on devices up to 7". The iOS, Windows Phone, and HTML5 versions are still under development [18].

Similar to Scratch, programs in Pocket Code are created by snapping together command blocks which are called "bricks" (see Fig. 3). The bricks are arranged in "scripts" which can run in parallel allowing concurrent execution. Broadcast messages are used to ensure sequential execution of scripts [6].

The language elements are organized into the categories "Event", "Control", "Motion", "Sound", "Looks", "Pen", "Data", "Lego EV3", "Phiro", and "Arduino". Each language element contains a set of bricks as shown below [19]:

Fig. 3 Example of Catroid program.

Event: Elements of this category are used to sense events, which trigger scripts to execute. Event's bricks are needful for every project without these bricks from this category, a project would not be able to start. By using bricks of this category broadcast messages can be sent and received, and events can be caught.

Control: Elements of this category are responsible for the program's flow, condition and loops can be defined.

Motion: Elements of this category are used to control an object's movement such as modify the position and the orientation of an on-screen object and can be used to set the gravity and mass for objects.

Sound: These elements play or stop a predefined sound file (sounds may also be recorded via the internal sound recorder) and alter the volume. Google's speak engine is used to read out some text.

Looks: The visual appearance of objects can be defined with these elements, the bricks are used to control an object's appearance. The size of an object can be set as well as transparency, brightness, and color. The elements of this category can be used to switch between the backgrounds. Also controlling some device peripherals is possible with elements of this category such as camera and flashlight.

Pen: The Pen category allows an object to draw shapes, plot colored pixels. The elements are used to change the size of the pen by a chosen number, set the size of the pen to a chosen number, and draw a copy of the object on the stage.

Data: Setting and changing the value of a variable is possible with elements of this category. The elements of this category also can be used to perform data manipulation operations over lists.

Lego EV3: Elements of this category are used to program Lego Mindstorms EV3 robot.

Phiro: Elements of this category can be used to program and control Phiro (educational robot) via Bluetooth.

6. Localization Catrobat into Bidirectional Languages

Smartphone apps localization has still not reached its full potential. One of the reasons why there are many smartphone's apps in App Stores are not localized is due to the developing rate of this field. Every day more and more apps loaded to the market, and companies which want to keep up do not enough consider internationalization and localization activities during the product development phase.

In this section, we provide all the challenging aspects of localization Catrobat into bidirectional languages and achieve comprehensive solutions to the many and correlated challenges presented by such languages. Software localizers understand the challenges of changing an LTR VPE into RTL. Therefore, we work carefully to comply with bidirectionality design guidelines and provide a consistent right-to-left look and feel product that suits the children needs and expectations. In particular, visually appealing, easy-to-use UI and graphics are our default objectives in localizing Catrobat into bidirectional languages.

6.1 Mirroring Bricks' Background

The direction of reading and writing influences how information should be drawn on the screen (i.e. mirroring awareness). Layout mirroring is a term used to describe the ability of an app to reorder the UI elements to match the right to left rendering for bidirectional locales. This requires, not only the text alignment and text reading order flow from right to left, but also the UI elements layout. The requirement for mirroring occurs since the text in such languages is read and written from right-to-left rather than left-to-right [12].

The background's graphic for each brick in Pocket Code communicates direction as shown in Fig. 4 (a). Hence, to provide a consistent right-to-left look and feel to the brick layout features, the background should be automatically mirrored when its layout direction is right-to-left. In particular, the brick's layout is displayed flowing from right to left (see Fig. (b)), and the following variations occur.
1. Curved area flows from RTL.
2. Touch target is aligned to the right.

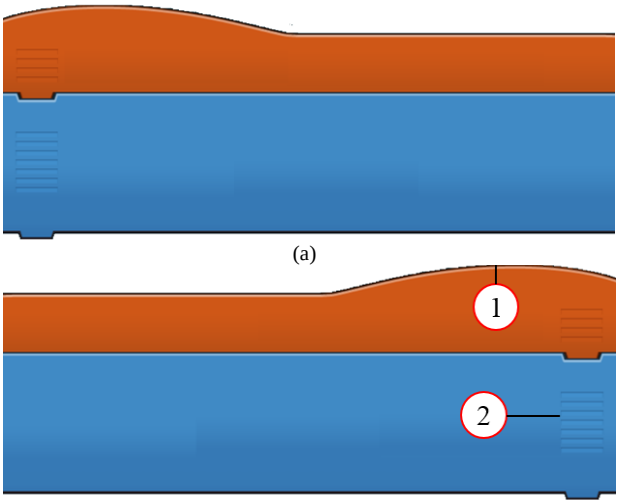


Fig. 4 Screenshots for bricks (a) LTR (b) RTL.

Enabling mirroring in resources: The background for each brick should reverse its horizontal orientation for right-to-left layouts; developers can include the mirrored images in a drawable-ldrtl/ resource directory. Particularly, in Pocket Code, the background for each brick is a 9-patch drawable. A 9-patch drawable enables the user to create bitmap images that automatically change the size to accommodate the views' contents and the size of the smartphone's screen [20]. The border is used to define the stretchable and static sections of the image. A stretchable section is defined by drawing one (or more) 1-pixel-wide black line(s) on the left and top part of the border (the other border pixels should be fully transparent or white) [20] as shown in Fig. 5. Therefore, due to these special attributes for the 9-patch drawable, the localization for drawable files (i.e. creating the custom version of drawable) in Pocket Code is an infeasible solution.

Fig. 5 Example of 9-patch drawable in Pocket Code.

Enabling mirroring in XML file: The (scaleX) property is used to scale of the view in the x direction. When X scale factor equals -1 (i.e. x values increase from right to left), UI element will be flipped horizontally. In XML file which defines the layout of brick, the (scaleX) property is configured as shown in the below XML code:

```
<org.catrobat.catroid.ui.BrickLayout
    android:id="@+id/brick_broadcast_layout"
    style="@style/BrickContainer.Control.Medium"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    app:horizontalSpacing="@dimen/brick_flow_layout_horizontal_spacing"
    app:verticalSpacing="@dimen/brick_flow_layout_vertical_spacing"
    android:scaleX="-1"
    >
```

The main drawback of this solution is that, for each implemented brick layout, localizers need to create an alternative resource file and place it within specially named subdirectories of the Pocket Code. Manually repeating this for each implemented or new brick would be error prone and a waste of time. To gain more flexibility, the second way - mirroring in source code - is used instead.

Enabling mirroring in source code: There is a bug in 9-patch drawable with auto-mirroring, where it breaks the transformation matrix for drawing the child view afterward. The challenging aspect here is to mirror the background for each brick without mirroring the UI elements. Therefore, the Java class file which defines the layout of brick should be modified. The source code enables the localizers to get the background only and then perform the mirroring technique. Practically, we need to detect the reverted matrix, reset it, mirror the background of brick and then draw it again.

The following snippet of code illustrates how the matrix is detected and reset, and the dispatchDraw () method shows how the mirroring and drawing methods are implemented. By keeping in mind that this method is used when the Android version is prior to KITKAT.

```
@Override
protected void onDraw(Canvas canvas) {
   float[] values = new float[10];
   canvas.getMatrix().getValues(values);
   if (values[0] < 0.0f)
      canvas.setMatrix(new Matrix());
   super.onDraw(canvas);
}
@Override
protected void dispatchDraw(Canvas canvas) {
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR1 &&
         Build.VERSION.SDK_INT < Build.VERSION_CODES.KITKAT &&
         getLayoutDirection() == LAYOUT_DIRECTION_RTL) {
      Drawable background = getBackground();
      int color = getResources()
            .getColor(R.color.application_background_color);
      canvas.drawColor(color);
      canvas.save();
      canvas.translate(background.getBounds()
            .right - background.getBounds().left, 0);
      canvas.scale(-1.0f, 1.0f);
      background.draw(canvas);
      canvas.restore();
   }
   super.dispatchDraw(canvas);
}
```

Android 4.4 (KITKAT) offers a new feature for drawable mirroring for RTL layout. In practice, the system can automatically mirror background images for all bricks by calling setAutoMirrored (). When the smartphone's language is set to Arabic, the backgrounds will be automatically mirrored - see Fig. 4 (b). The below snippet of code illustrates how we can mirror the brick's background using the setAutoMirrored () method.

```
protected void onFinishInflate() {
   super.onFinishInflate();
   Drawable background = getBackground();
   if (background != null) {
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
         background.setAutoMirrored(true);
      }
   }
}
```

6.2 Reordering View's Children

The Android framework supports several default views but a developer can also create their own custom views and call them in their application. Views are typically created to provide high user-interface capabilities and experience which are not possible with the built-in views.

In Android, a view group is a special view that can hold other views. Sometimes, because of specific nature of requirements, the standard layout managers are not enough or developers are not satisfied with the existing functionality of any of the available layout managers.

Therefore, developers need to extend the view group class to create their own custom layout manager to suit their needs [21].

Views are capable to measure, lay out and draw themselves and their child elements (in a case of a view group). Views are also providing the ability to save their UI state and handle touch events [21]. Parents are drawn before their children and children are drawn depending on the order when they are called. Drawing custom layout manager is done by passing three processes: measure, layout and draw as shown in the below Fig. 6:

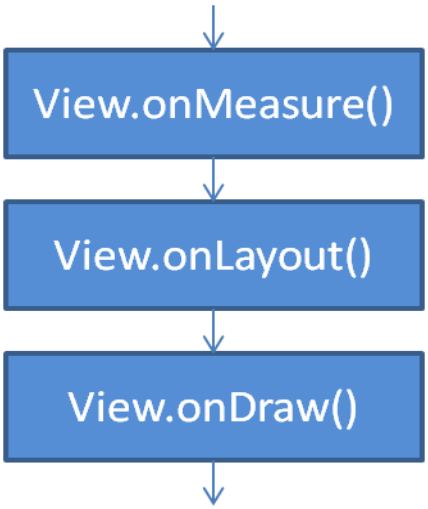


Fig. 6 The three processes for drawing custom layout manager.

For all these features seen so far, in Pocket Code, the brick layout is implemented as a custom view since various design experience and functionalities are needed as shown in Fig. 7 (a).

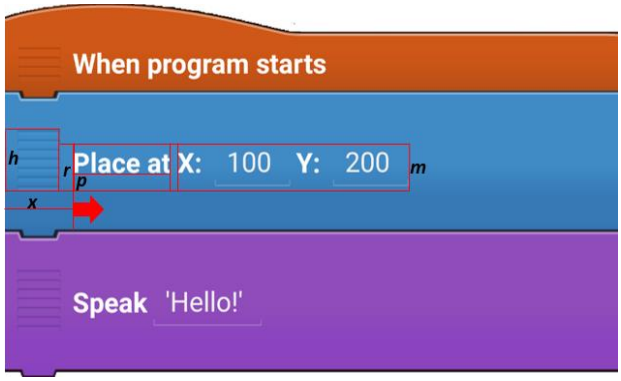
User interface design is the most critical issue in bidirectional languages app. However, when designing for children and young people, developers often think that highly visual interface, simple interaction is the way to grab their interest [22]. For bidirectional languages, not only does the text alignment and text reading order rendered from right to left, but also the UI elements layout should follow this natural direction of rendering. Of course, this layout change would only apply to localized bidirectional languages.

When mirroring the brick's layout, padding and margin around UI elements and text also switch placement to match RTL layouts. However, in Fig. 7 (b), the text view for "Place at" is aligned to the right, and the size and position requirements for the view and all of its children should match the LTR version (see Fig. 7 (a)) as follow:

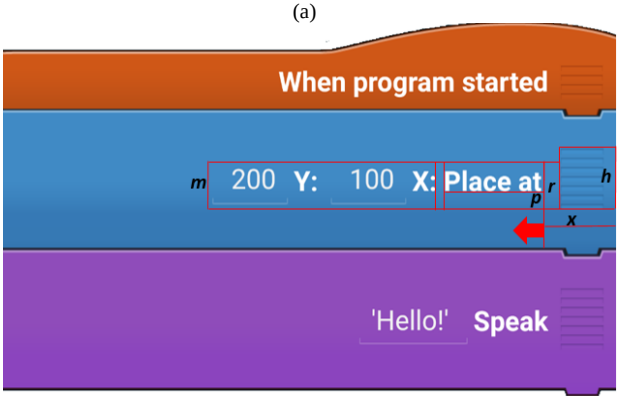
- Touch target height: (h) dp
- Screen edge margin before first UI element: (x) dp
- Text view bottom padding: (p) dp
- Text view left padding: (r) dp
- UI elements height: (m) dp

However, the onMeasure () method should be customized to introduce new measurements that support child laying out from right-to-left and adjust the brick's children alignment to the right. More specifically, the variable (posX) that determines where is to start laying out should be modified to handle both modes (LTR and RTL) based on mobile device's locale as shown in the below snippet of code.

```
if (config.getLayoutDirection() == View.LAYOUT_DIRECTION_RTL) {
   posX = sizeWidth - lineLengthWithHorizontalSpacing;
} else {
   posX = getPaddingLeft() + lineLength - childWidth;
}
```



(a)



(b)

Fig. 7 LTR vs. RTL laying out for bricks.

In onLayout () we need to call layout method on each child of this view group and provide desired position (relatively to parent) for them using the sizes computed in the onMeasure () as shown in the below snippet of code. However, Fig.8 shows the Arabic localized version for Fig.3.

```
@Override
protected void onLayout(boolean changed, int left,
                int top, int right, int bottom) {
   final int count = getChildCount();
   for (int i = 0; i < count; i++) {
      View child = getChildAt(i);
      LayoutParams layoutParams = (LayoutParams) child.getLayoutParams();
      child.layout(layoutParams.positionX, layoutParams.positionY,
            layoutParams.positionX + child.getMeasuredWidth(),
            layoutParams.positionY + child.getMeasuredHeight());
   }
}
```

Fig. 8 Example of localized BiDi program.

6.3 Bidirectional Characters Rendering

Bidirectional text such as Arabic requires special rendering since there are different characteristics of scripts rendering include bi-direction, shaping as per context, reordering and linking characters [12, 17].

In Pocket Code, you can display text containing English, German, French, and Spanish string all at once. But, there are several scripts that need special processing to render and edit since the characters are not laid out in a simple linear progression from left to right, as most European scripts are. These writing systems are referred to as “complex scripts.”

To investigate more in this issue, a program which contains bricks for displaying text on the program’s stage is created. In Fig. 9, the variable (text1) is initialized to the English text, while the variable (text2) is initialized to Arabic. However, when the program is executed, the stage correctly displays the whole English text and oddly does not display any Arabic text as shown in Fig. 10 (a). That is because Pocket Code, unfortunately, does not support bidirectional characters rendering on the stage and deliver its own font. Therefore, to make Pocket Code work properly for complex scripts when displaying typed text, the characters should not be outputted one at a time, the text should be saved in a buffer and then displayed as a whole on the screen.

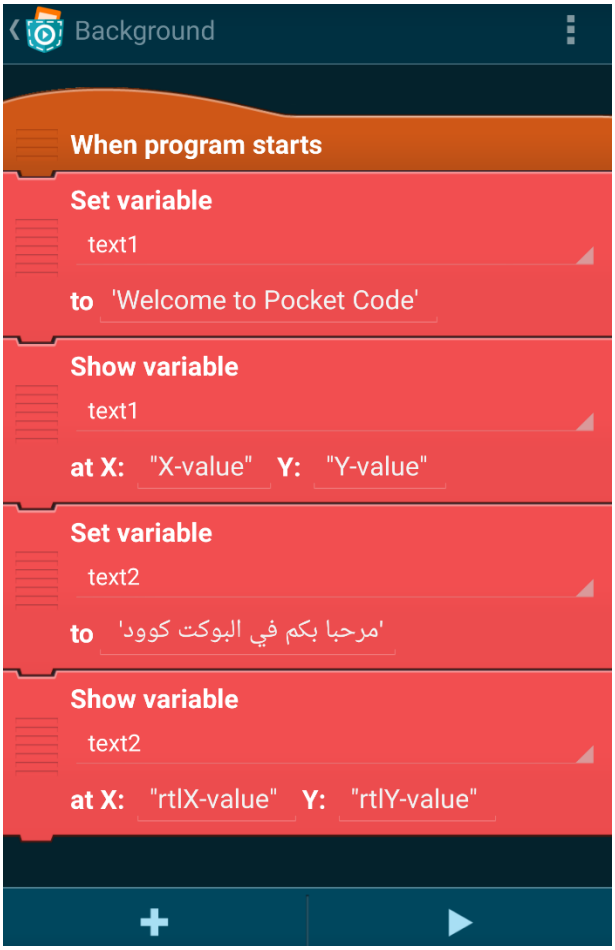
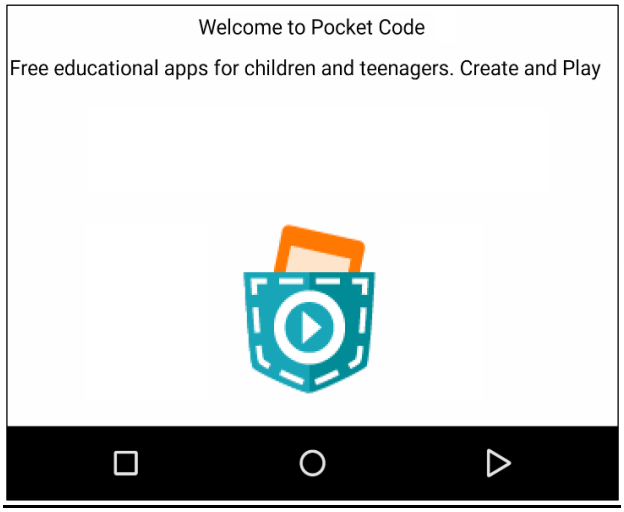
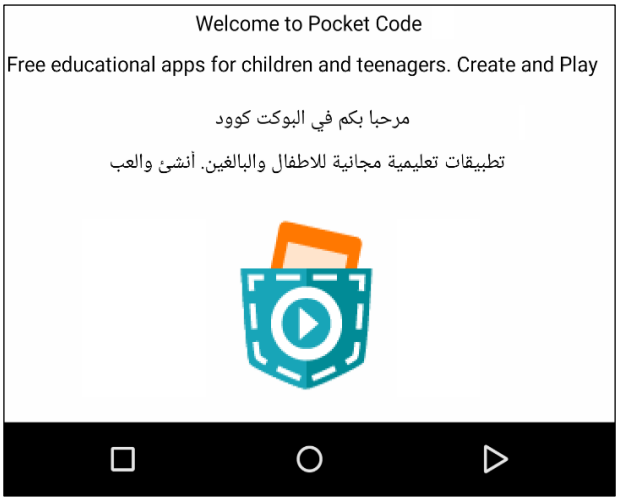


Fig. 9 Screenshot for the set and show variable bricks with English and Arabic strings.



(a)



(b)

Fig. 10 Screenshots for Pocket Code's stage (a) displays Latin scripts (b) displays Latin and BiDi characters.

In Pocket Code, the bitmap font is used to draw characters on the stage. This technique does not support displaying of complex scripts. When language scripts are rendered from right to left, we might have to use other techniques than just drawing characters next to each other on the screen to produce something intelligible. In particular, to support complex bitmaps font rendering such as Arabic, Urdu, Persian, etc., we need some way to convert the bitmap image that contains the text into texture object [23]. However, we use a single texture object to render all the glyphs. In order to draw the bidirectional text, we load the characters, upload them as a texture, and draw them at the correct offset from the starting position. When smartphone is set to the Arabic language, the Arabic version of Pocket Code correctly displays the text on the stage with proper morphology and directionality as shown in Figure 10 (b).

6.4 Bidirectional User Interface Mirroring

One of the most localization challenging issues is that some UI elements are being drawn outside the desired area or outside the UI canvas in localized version because its coordinates are expecting the origin to be in the top-left of the screen instead of the top-right. In Pocket Code, when the Arabic user tries to show the details of sprites, no details will be displayed on the screen. This defect due to the localization process, all the views' positions in Fig. 11 (a) are changed to be drawn outside the screen and no details will be shown.

To support RTL mode, we need to provide some specific details view layout for any bidirectional languages, the custom version of the layout is created and placed within specially named subdirectories of the Pocket Code. Some code snippet is needed to adjust and correctly reorder the positions of text views by using the (*layout_toLeftOf*) instead of (*layout_toRightOf*). Further, we use "*start*" and "*end*" instead of "*left*" and "*right*" in style file to provide some specific styles for the BiDi languages. Also, for more precise control over the app UI in RTL mode, the attribute for setting the direction of a components' text is used: *textDirection="rtl"*.

In Fig. 11(b), title, icons, and UI elements are displayed flowing from right to left, however, when a UI layout is mirrored, these variations occur:

1. Back button points to the right.
2. Arabic text is right-aligned.
3. Menu button is left-aligned.
4. Icon appears to the right of text.
5. Scripts' field appears to the right of value.
6. Play button is not mirrored since it reflects the direction of the tape.

6.5 Generating Strings during Application Runtime

In the course of localizing software, all strings should be declared as resources and separated from the source code [8]. Typically, program size is expressed in units of measurement (KB, MB, GB, etc.). On projects' details screen (see Fig.12 (a)) these units are implemented as a hard-coded string since the size of each program should be computed, and then mapped to the proper unit of measurement at the program's run time. On an RTL screen (see Fig. 12 (b)), the placement of unit ("MB") appears to the left of value, further, the following localization issues should be considered:

1. Size's value is generated at the run time.
2. "MB" unit is a hard-coding string (i.e. untranslated).
3. Hindi digits are used for numbering style.

(a) (b)

Fig. 11 Screenshots for sprite-details screen (a) original version (b) RTL screen.

The developer has to be careful with strings that are composed at the application's runtime. To solve this issue, all of the strings should be moved into resource file; the content of the file can be retrieved by the software at the run time to supply these elements to the users in their local language. The following snippet of code is used to remove the hard-coding string of measurement's units.

```
public static String formatFileSize(long bytes, Context context) {
   final double unit = 1024;
   String fileSizeExtension[] = new String[]{
         context.getString(R.string.Byte_short),
         context.getString(R.string.kiloByte_short),
         context.getString(R.string.kiloByte_short),
         context.getString(R.string.GigaByte_short),
         context.getString(R.string.TeraByte_short),
         context.getString(R.string.PetaByte_short),
         context.getString(R.string.ExaByte_short)
   };
   if (bytes < unit) {
      return bytes + " " + fileSizeExtension[0];
   }
   int exponent = (int) (Math.log(bytes) / Math.log(unit));
   exponent = Math.min(exponent, fileSizeExtension.length - 1);
   String prefix = fileSizeExtension[exponent];
   return String.format(Locale.getDefault(), "%.1f %s", bytes /
         Math.pow(unit, exponent), prefix);
}
```

(a)

(b)

Fig.12 Screenshots for program-details screen (a) original version (b) RTL screen.

7. Analysis and Results

After the product localization, the bidirectional languages localization testing framework, which was proposed by Ayyal Awwad and Slany [8], is performed. The objective is to ensure that the localized version of Pocket Code is fully functional, cosmetically correct, linguistically accurate, and culturally appropriate and that no issues have been produced during the localization process.

Primarily, to verify whether the target VPE meets the localization requirements or not, we need to check the product in terms of complying with Material Design guidelines for bidirectionality. Furthermore, during the localization phase, translation may cause some GUI defects (e.g. truncated strings, overlapping controls, and misalignment). These defects should be considered during the testing process as illustrated in Table 3.

Table 3: Bidirectional languages testing methods

Testing Methods Description	***Result***
Localized strings validation (i.e. user-facing text should be bidirectional scripts)	passed
Mirroring awareness: UI elements lay out from RTL	passed
Text direction (text reading order go from RTL)	passed
Complex scripts awareness (app render and edit bidirectional scripts)	passed
Direction-sensitive graphics (such as undo and redo)	passed
String truncation validation	passed
UI overlapping validation	passed
Views alignment (i.e. right-aligned elements)	passed
Linguistic verification: check the semantic correctness of the translations using product's screenshots	passed
Images and colors: issues of cultural appropriateness	passed

The test methods are executed on the bidirectional languages which are used as the reference. The methods had been created to perform a round trip and visit every screen, dialog, and menu. The test cases methods take screenshots of the visited screens and extract text strings for analysis purposes.

From Table 3, we can observe that Pocket Code complies with the design guidelines for bidirectionality and not only does the text alignment and text reading order rendered from right to left, but also the UI elements layout follow this natural direction.

For cultural appropriateness testing, all of the application's graphics should be culture-neutral, testing is performed at two levels. Each individual view is tested and each screen or layout is tested as a whole. For usability testing, two representatives of each culture participate. The participants check each view to determine whether or not there are any peculiarities of that view that might be offensive or include any attributes that use the idioms of a specific culture. If anything is found unsatisfactory, that view should be customized using the localization tool to be satisfactory.

8. Conclusion

In this paper, we localized a VPE on smartphones into bidirectional languages and presented the challenging aspects of the bidirectional languages (specifically Arabic language) facing the software localization team. These challenges stem from the fact that these languages differ enormously in terms of their characters, contextual shaping, and directionality from Western languages [17].

Our software localization process includes adapting Catrobat to be used in the education system in primary and secondary schools. That process means changing, for example, custom and standard layout directions, text rendering and locale formats to suit the new culture it is being localized to. Our localization process also includes changing the source code of Catrobat and in many cases localizing icons to suit the local users' needs and expectations. The localization testing results show that the product meets bidirectional requirements and complies with bidirectionality design guidelines since it is cosmetically correct, linguistically accurate, and culturally appropriate.

9. Future Work

There are some correlated challenging aspects for localization of the major Asian languages (Chinese, Korean, and Japanese) [15]. In particular, the peculiarities of Japanese and Korean languages raise some issues during the translation process for Catrobat. These challenges stem from the fact that the verb comes last in such languages as well as plural and singular are the same.

Our Future work is to propose a comprehensive solution that is based on the idea of implementing all strings with the placeholder for the parameters. This would allow translators to better understand the context and decide at which place the parameters should appear in the localized string. However, the Android string resource system does not support parameters, nor does localization management platform.

Acknowledgments
The author would like to thank Professor Wolfgang Slany and the Catrobat development team[1].

References

[1] P. Smutný, "Visual Programming for Smartphones", 12th International Carpathian Control Conference (ICCC), 2011, pp. 358-361.
[2] H. Tsukamoto et al., "Textual vs. Visual Programming Languages in Programming Education for Primary Schoolchildren", IEEE Frontiers in Education Conference (FIE), 2016, pp. 1-7.
[3] Wolfgang Slany, "A Mobile Visual Programming System for Android Smartphones and Tablets", IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2012, pp. 265-266.
[4] Wolfgang Slany, Catrobat, http://no1leftbehind.eu/pocket-code/, 2010-2017.
[5] Wolfgang Slany, Catrobat, http://developer.catrobat.org/. 2010-2017.
[6] Wolfgang Slany, "Catroid: A Mobile Visual Programming System for Children", Proceedings of the 11th International Conference on Interaction Design and Children, 2012, pp. 300-303.
[7] C. Williams, E. Alafghani, A. Daley, K. Gregory and M. Rydzewski, "Teaching Programming Concepts to Elementary Students", IEEE Frontiers in Education Conference (FIE), 2015, pp. 1-9.
[8] Aiman M. Ayyal Awwad, Wolfgang Slany, "Automated Bidirectional Languages Localization Testing for Android Apps with Rich GUI", Mobile Information Systems, Vol. 2016.
[9] Google, Material guidelines: Usability-Bidirectionality, https://material.io/guidelines/usability/bidirectionality.html. (visited on 10 April 2017).
[10] X. Xia, D. Lo, F. Zhu, X. Wang and B. Zhou, "Software Internationalization and Localization: An Industrial Experience", 18th International Conference on Engineering of Complex Computer Systems, 2013, pp. 222-231.
[11] Admob; Google, https://www.statista.com/statistics/296304/mobile-app-abandoment-rate-due-to-lacking-localization/, (visited on 10 January 2017).
[12] S. Abufardeh, K. Magel, "QA/Testing Bi-directional Languages Software: Issues and Challenges", 32nd Annual IEEE International Computer Software and Applications Conference, 2008, pp. 172-175.
[13] Elvis Hau, Manuela Aparício, "Software Internationalization and Localization in Web Based ERP", Proceedings of the 26th Annual International Conference on Design of Communication, "SIGDOC, 2008, pp. 175-180.
[14] Nations Online, Countries and Languages, http://www.nationsonline.org/oneworld/countries_by_languages.htm, (visited on 20 January 2017).
[15] S.Abufardeh, K.Magel, "Software Localization: the Challenging Aspects of Arabic to the Localization Process (Arabization)", in Proceedings of the IASTED International Conference on Software Engineering (SE'08), 2008, pp. 275-279.
[16] Arwa Alqudsi, Nazlia Omar, Khalid Shaker, "Arabic Machine Translation: a Survey", Artificial Intelligence Review, Vol. 42, No. 4, 2014, pp. 549-572.
[17] T. A. El-Sadany, M. A. Hashish, "An Arabic Morphological System", IBM Systems Journal, Vol. 28, No. 4, 1989, pp. 600-612.
[18] Wolfgang Slany, "Pocket Code: a Scratch-Like Integrated Development Environment for Your Phone", ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity, 2014, pp. 35-36.
[19] Wolfgang Slany, Catrobat education, https://edu.catrob.at/brick-documentation. 2010-2017.
[20] Android Developers, Create Resizable Bitmaps (9-Patch files), https://developer.android.com/studio/write/draw9patch.html, (visited on 10 December 2016).
[21] Android Developers, View, https://developer.android.com/reference/android/view/View.html (visited on 21 January 2017).
[22] B. DiSalvo, "Graphical Qualities of Educational Technology: Using Drag-and-Drop and Text-Based Programs for Introductory Computer Science", IEEE Computer Graphics and Applications, Vol. 34, No. 6, 2014, pp. 12-15.
[23] OpenGL, Texture Mapping, https://www.opengl.org/archives/resources/faq/technical/texture.htm, (visited on 2 February 2017).

[1] http://developer.catrobat.org/credits

Aiman Mamdouh Ayyal Awwad is currently pursuing a Ph.D. in Computer Science with research interests related to smartphone applications. He received his B.Sc in Computer Science from Mutah University in 2007 and his M.Sc in Computer Science from University of Jordan in 2010. From February 2010 to September 2014, he was a lecturer at Computer Science and IT Department/ Tafila Technical University. He has more than 6 publications in various international journals and conferences. His research interests include mobile applications testing, image processing, and cellular automata.