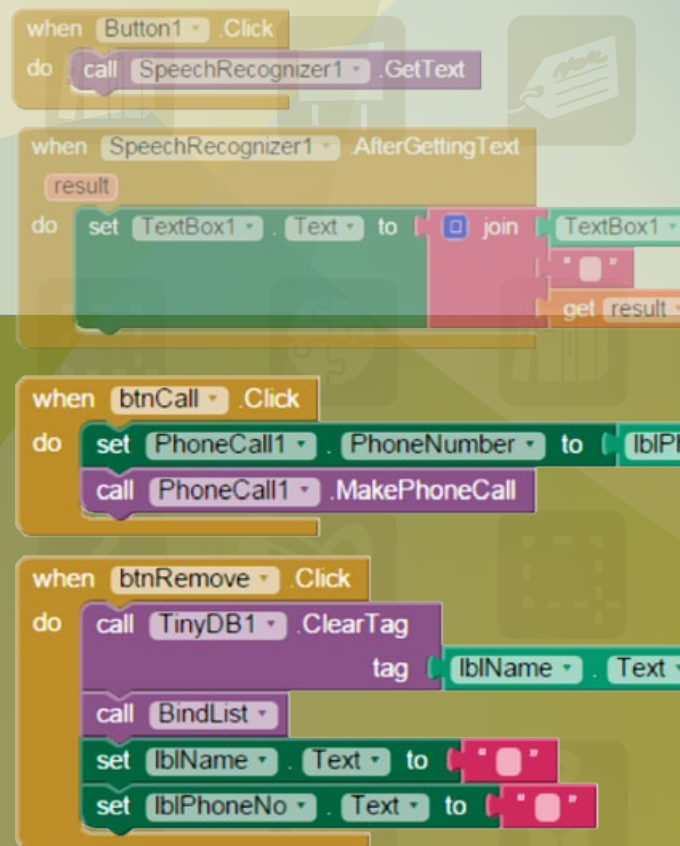




COMMONWEALTH of LEARNING

Educational App Development Toolkit for Teachers and Learners



Educational App Development Toolkit for Teachers and Learners

Ishan Sudeera Abeywardena



COMMONWEALTH *of* LEARNING

The Commonwealth of Learning (COL) is an intergovernmental organisation created by Commonwealth Heads of Government to promote the development and sharing of open learning and distance education knowledge, resources and technologies.



Commonwealth of Learning, 2015

© 2015 by the Commonwealth of Learning. *Educational App Development Toolkit for Teachers and Learners* is made available under a Creative Commons Attribution-ShareAlike 4.0 Licence (international): <http://creativecommons.org/licenses/by-sa/4.0>.

For the avoidance of doubt, by applying this licence the Commonwealth of Learning does not waive any privileges or immunities from claims that it may be entitled to assert, nor does the Commonwealth of Learning submit itself to the jurisdiction, courts, legal processes or laws of any jurisdiction.

Educational App Development Toolkit for Teachers and Learners

Ishan Sudeera Abeywardena, PhD

ISBN: 978-1-894975-73-5

Image credits: pp. 126-127.

Published by:

COMMONWEALTH OF LEARNING
4710 Kingsway, Suite 2500
Burnaby, British Columbia Canada V5H 4M2

Telephone: +1 604 775 8200

Fax: +1 604 775 8210

Web: www.col.org

Email: info@col.org

Contents

About the Toolkit	1
How to Use the Toolkit	2
Using the Toolkit: Students	2
Using the Toolkit: Teachers	3
1. MIT App Inventor	4
1.1 Introduction	4
1.2 Creating an App Inventor Account	5
1.3 The Development Environment	5
1.4 Device Setup for App Development and Debugging	7
1.4.1 Building Apps with an Android Device and Wi-Fi Connection	8
1.4.2 Building Apps with the Emulator	11
1.4.3 Building Apps with an Android Device and USB Cable	14
2. Activities	19
Activity 1: TextToSpeech (Loud Mouth)	21
Activity 2: AccelerometerSensor (Stop Shaking Me)	31
Activity 3: SpeechRecognizer (Dictation)	35
Activity 4: Canvas (Join the Dots)	40
Activity 5: Ball and ImageSprite (Move the Ball)	43
Activity 6: TinyDB, PhoneNumberPicker and PhoneCall (Call a Friend)	51
Activity 7: Camera (Say Cheese!)	61
Activity 8: Camcoder and VideoPlayer (Action Capture)	64
3. Tutorials	67
Tutorial 1: Pet the Kitty	67
Tutorial 2: Swat the Mosquito	70
Tutorial 3: Dice	73
Tutorial 4: Virtual Chemistry Experiments	77
Tutorial 5: Scan and Learn	84
Tutorial 6: Voice Note	87
4. Packaging and Distribution	96
4.1 Sharing your app so that others can remix it (.aia file)	96
4.2 Sharing your app for others to install on their phone/tablet (.apk file)	97

5. Publishing Apps on Google Play	99
5.1 The Developer Console	99
5.1.1 All Applications	99
5.1.2 Account Details	100
5.1.3 Linking Your Merchant Account	101
5.1.4 Multiple User Accounts	101
5.1.5 Store Listing Details	102
5.1.6 Upload and Instantly Publish	104
5.1.7 Alpha- and Beta- Testing	104
5.1.8 Staged Rollouts	105
5.2 Multiple APK Support	105
5.3 Selling and Pricing Your Products	106
5.4 In-App Products	107
5.4.1 Distribution Controls	107
5.4.2 Geographic Targeting	107
5.4.3 Capabilities Targeting	107
5.5 User Reviews and Crash Reports	109
5.5.1 App Statistics	110
6. Workshop Design and Evaluation	111
6.1 Workshop Training Schedule	111
6.2 Workshop Evaluation	114
7. Useful Resources	119

LIST OF FIGURES

Figure 1.1 The Designer view of the AI2 platform	6
Figure 1.2 The Blocks view of the AI2 platform	6
Figure 1.4 QR code to download the Companion app from the Google Play Store	8
Figure 1.3 Building apps with an Android device and Wi-Fi connection	8
Figure 1.5 Connecting to AI Companion from AI2	9
Figure 1.6 Connecting to your project using a QR code or six-character code	10
Figure 1.7 Using the emulator to build apps	11
Figure 1.8 The aiStarter icon for Windows	12
Figure 1.9 aiStarter start-up window	12
Figure 1.10 Connecting to the emulator from AI2	13
Figure 1.11 Update screens during emulator start-up	13
Figure 1.12 The four phases of emulator start-up	14

Figure 1.13 Using a USB cable to build apps	14
Figure 1.14 QR code to download the Companion app from the Google Play Store	16
Figure 1.15 The aiStarter icon for Windows	16
Figure 1.16 aiStarter start-up window	17
Figure 2.1 Using the Palette, Viewer and Properties in Designer view	19
Figure 2.2 Using the Blocks drawers in Blocks view	20
Figure 2.3 Click on the Start new project button to open a new AI2 project	22
Figure 2.4 Name your project using the project name provided in the activity sheet	23
Figure 2.5 The project name is displayed inside the AI2 project	23
Figure 2.6 Blank phone screen used to develop the visual aspects of the app	24
Figure 2.8 Change the title of the app	25
Figure 2.9 Locate the TextBox and Button components	25
Figure 2.10 Drag and drop TextBox and Button onto the blank phone screen	25
Figure 2.11 TextToSpeech component located within the Media drawer of the Palette	26
Figure 2.12 Non-visible components shown outside the visible area of the phone screen	26
Figure 2.13 Change the Hint property of the TextBox component	27
Figure 2.14 Change the Text property of the button to “Speak”	27
Figure 2.15 Use the Designer and Blocks buttons to change between the Designer view and the Blocks view	28
Figure 2.16 The logic block for the LoudMouth app	28
Figure 2.17 Click on Button1 in the Blocks section and drag and drop the when Button1. Click do block into the empty space.	29
Figure 2.18 Join the two blocks so they click together	30
Figure 2.19 Join all three blocks to make the logic of your app	30
Figure 2.20 More information popup for the TextToSpeech component	33
Figure 2.21 The logic block for the Stop Shaking Me app	33
Figure 2.22 Using an empty Text block	34
Figure 2.23 Upload an image from the Resources pack	37
Figure 2.24 The first logic block, which captures the user’s speech	38
Figure 2.25 Appending text to the existing text when the user has stopped speaking	38
Figure 2.26 Adding and deleting grooves in a join block	39
Figure 2.27 Getting the result in a SpeechRecognizer.AfterGettingText block	39
Figure 2.28 Creating a new procedure	47
Figure 2.29 Renaming a procedure	47
Figure 2.30 Procedure for randomly setting the positions of the Ball and the Hole	47
Figure 2.31 Using the random integer block in the Math section	48
Figure 2.32 Calling a procedure at Screen Initialize	48

Figure 2.33	The block to determine what happens when the Ball is flung by the user	48
Figure 2.34	Moving the Ball back inside the bounds of the screen	49
Figure 2.35	Making the Ball bounce back off of the edge of the screen	49
Figure 2.36	Detecting a collision between the Ball and the ImageSprite	50
Figure 2.37	The VerticalArrangement component	55
Figure 2.38	Properties of the VerticalArrangement component	55
Figure 2.39	The HorizontalArrangement component	55
Figure 2.40	Renaming a component	57
Figure 2.41	Using variables to store and display a value	57
Figure 2.42	Inserting a Tag-Value pair into a TinyDB	59
Figure 2.43	Inserting a contact from the PhoneNumberPicker into a TinyDB	59
Figure 2.44	Removing a Tag-Value pair from the TinyDB	60
Figure 2.45	Populating the ListPicker with the Tags in the TinyDB	60
Figure 4.1	Exporting a project as an .aia file	96
Figure 4.2	Importing a project into AI2	97
Figure 4.3	Packaging an app as an .apk file	97
Figure 4.4	Build progress of the .apk file	98
Figure 5.1	Google Play Developer Console	99
Figure 5.2	All applications on the Google Play Developer Console	100
Figure 5.3	Google Developer account details	101
Figure 5.4	Multiple user account in the Developer Console	102
Figure 5.5	Google Play Store listing details	103
Figure 5.6	Alpha- and beta-testing your app	104
Figure 5.7	Selling and pricing products on Google Play	106
Figure 5.8	Device compatibility	108
Figure 5.9	User reviews and ratings of your app	109
Figure 5.10	Detailed statistics about your app	110

About the Toolkit

The Android operating system (OS) is arguably dominating the current smartphone and tablet market. The Free and Open Source Software (FOSS) frameworks and ease of use have made it the most sought-after OS for use by manufacturers. With thousands of apps available through the Google Play Store, Android provides a feature-rich experience with an app for just about anything imaginable.

The exponential growth of the smartphone and tablet markets over the past few years has caught the attention of many sectors including government, industry and educators. Mobile Business (mBusiness), Mobile Learning (mLearning) and Mobile Government (mGovernment) are just three of the fastest-growing sectors delivering information and services to a global market through mobile devices. This has given rise to a massive demand for various customised apps, which has in turn resulted in businesses and services investing heavily in custom mobile applications.

Traditionally, Android app development has been a highly specialised field reserved only for software engineers and programmers. However, the massive demand for customised apps has led to the democratisation of Android app development through Visual Programming. Visual Programming is a concept that allows non-programmers to build powerful applications using logical building blocks — like constructing a jigsaw puzzle from virtual puzzle pieces. Each puzzle piece is a block of code that forms complete programs when assembled logically. The current leading visual app development platform is App Inventor (AI2), developed by the Massachusetts Institute of Technology (MIT). It harnesses the power of Google to provide a robust solution for customised app development for the Android OS.

This toolkit is designed to introduce the basic and some intermediate concepts of Android app development on the AI2 platform. By following the hands-on activities and tutorials, you will learn to use the Designer and Blocks Editor components of App Inventor to create apps that can be readily downloaded and used on your Android smartphone or tablet. Furthermore, the hands-on sessions will guide you on how to use features such as text-to-speech, accelerometer, speech recognition, drawing, video, games and music playback. You will also create six intermediate applications in the tutorials, which will give you a solid foundation for developing more complex apps in the future. Finally, you will learn how to package and distribute your app.

How to Use the Toolkit

This toolkit is designed to be used by both teachers (trainers) and students (trainees) who have little or no prior experience in software programming. It covers all aspects of Android application development on the AI2 platform, from account and device setup to publishing your app in the Google Play Store. You will be developing Android apps in two main categories. The first is Activities, relatively simple applications designed to give you hands-on experience in the use of one or two components at a time. By following the steps in the Walkthrough of each Activity, you will be able to complete these apps with relative ease. The second category is Tutorials, where you will be exposed to apps of increasing complexity built using multiple related and non-related components. You will also learn how to build complex logic blocks to achieve high levels of user experience (UX) within your app. There is no Walkthrough for the tutorials. However, as you should have completed all the activities prior to the Tutorials, you should have minimum difficulty in the development of the apps. Once you have completed all the sections in this toolkit, you will be able to develop Android apps for various purposes including commercial ones.

The toolkit acts as a resource for training of trainers (TOT). If you are a teacher who wants to teach Android app development, you must first work through this toolkit as a student. Once you have mastered app development on the AI2 platform, you can use the toolkit to conduct workshops to train others. Furthermore, since this toolkit document and the project resources are released under a Creative Commons Attribution-ShareAlike (CC BY-SA) 4.0 International licence, you are free to use them as you wish, even for commercial purposes.

Using the Toolkit: Students

- **Step 1:** Follow the instructions given in Section 1 to set up your AI2 account with live testing.
- **Step 2:** Familiarise yourself with the AI2 Design and Blocks views used for development.
- **Step 3:** Do all the activities in Section 2. Ensure that you build your apps sequentially from Activity 1 to Activity 8. Each activity is designed to build on the previous activities, so do not skip any.
- **Step 4:** Work through all six tutorials in Section 3. They are based on the knowledge you gained from doing the activities. Tutorials 1 to 6 are designed to take you from simple applications to intermediate and complicated application development. Each tutorial builds on the previous one, so follow the tutorials sequentially.
- **Step 5:** Practise packaging and distributing your apps as described in Section 4.
- **Step 6:** Familiarise yourself with the Google Play Developer Console and publishing apps on Google Play as discussed in Section 5.
- Once you have mastered app development on the AI2 platform, you are ready to teach others.

Using the Toolkit: Teachers

- Sections 1 to 5 are designed to be printed and distributed as individual handouts.
- All activities and tutorials are designed to be independent of each other. They can be printed and distributed to your students during the workshop.
- A workshop training schedule is provided in Section 6.1. All the workshop sessions are included in this toolkit. Each handout can be used as reference material when you conduct your workshop.
- A workshop evaluation form is included in Section 6.2. This can be used to gather feedback on your workshop for further improvement.
- The workshop must be conducted in a computer lab. All sessions are fully hands-on. Students should be encouraged to bring their own device for live testing. If this is not possible, they can use the emulator.

1. MIT App Inventor

1.1 Introduction¹

MIT App Inventor is a blocks-based programming tool that allows even novices to start programming and building fully functional apps for Android devices. Newcomers to App Inventor can have their first app developed and running in less than an hour, and can program more complex apps in significantly less time than with more traditional, text-based languages. Initially developed by Professor Hal Abelson and a team from Google Education while Abelson was on sabbatical at Google, App Inventor runs as a Web service administered by staff at MIT's Center for Mobile Learning — a collaboration of MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) and the MIT Media Lab. MIT App Inventor supports a worldwide community of nearly 3 million users representing 195 countries. More than 100,000 active weekly users have built more than 7 million Android apps between them.

The work of the MIT App Inventor team is driven by five primary objectives:

- **Sustaining and enhancing the tool:** A long-term commitment to sustaining and enhancing MIT App Inventor as a cutting-edge free service to end-users. To this end, the App Inventor team is continuously improving the tool, adding new features, debugging and enhancing its performance.
- **Building enterprise enhancements:** Work with public agencies and private corporations to support unique applications of the tool by developing or enhancing custom features of MIT App Inventor in response to partner needs.
- **Building capacity:** Seek to expand the capacity of formal and informal computing education for adults and youth around the world. In doing so, be actively engaged in developing and disseminating resources and training materials to support those interested in programming in their region.
- **Promoting computer science education:** Commit to calling attention to the state of computer science and computational thinking in education. From a policy perspective, participate actively in local and national conversations about standards of computing education; from an awareness perspective, participate in and support large-scale campaigns that aim to raise awareness among new audiences.
- **Conducting and supporting community research:** Undergraduate and graduate students at MIT and collaborating institutions are actively engaged in conducting and publishing research while developing, testing and evaluating the use of MIT App Inventor around the world.

¹ 1.1 Introduction is adapted from <http://appinventor.mit.edu/explore/about-us.html> (8 March 2015) under a CC BY-SA 3.0 Unported Licence.

As an open source tool that seeks to make both programming and app creation accessible to a wide range of audiences, MIT App Inventor has attracted the attention of:

- **Formal and informal educators** who have used MIT App Inventor to introduce programming to their Computer Science students, science club members, after-school programmes attendees and summer camp attendees. Many educators have also started to use MIT App Inventor to develop apps to support their own instructional objectives.
- **Government and civic employees and volunteers** who have harnessed the power of MIT App Inventor to develop often hyper-local apps in response to natural disasters and community-based needs.
- **Designers and product managers** who have seen the potential of MIT App Inventor to support the iterative design process via rapid prototyping, testing and iteration.
- **Researchers** who use MIT App Inventor to create custom apps to meet their data collection and analysis requirements to support their research in a wide variety of fields from, medical to social science.
- **Hobbyists and entrepreneurs** who have an idea they want to quickly turn into an app without the cost or learning curve that more traditional app creation entails.

1.2 Creating an App Inventor Account

To get started, you will need to create an App Inventor account. App Inventor 2 uses Google credentials for authentication, so you will first need to create a Gmail account if you don't already have one. To create a new Gmail account visit <https://www.gmail.com> and follow the instructions under **Create an Account**. Once you have created your Gmail account, visit <http://ai2.appinventor.mit.edu> to create your AI2 account. Use your Gmail address and password to sign in. You will then be directed to the Google accounts permissions page. To access the AI2 platform you need to click **Allow**. Congratulations! You are now ready to develop Android apps.

1.3 The Development Environment²

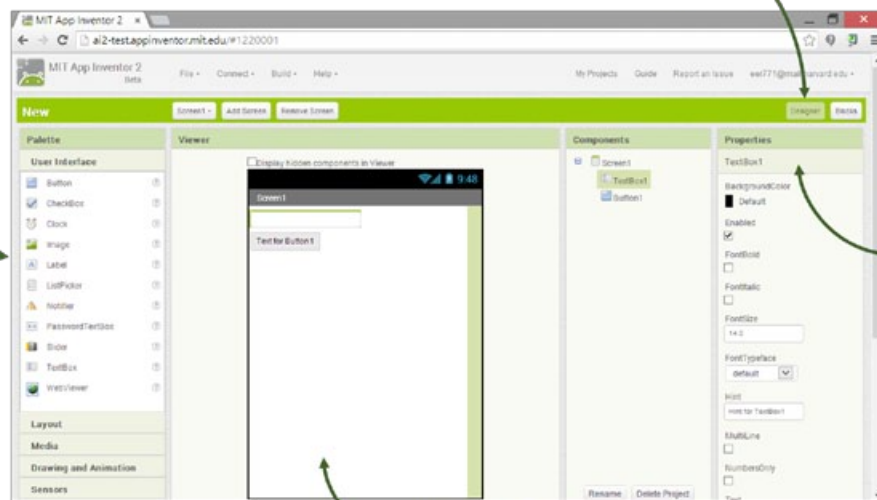
The app development on AI2 is twofold. The **Designer** view (Figure 1.1) gives you a visual representation of the application you are building. The **Blocks** view (Figure 1.2) provides a space for building the application logic using various logic components called **Blocks**. You will learn how to use these Blocks in the Activities.

² DesignTab.png and BlocksTab.png from <http://appinventor.mit.edu/explore/designer-blocks.html> are used under a Creative Commons Attribution-ShareAlike 3.0 Unported Licence.

Palette: Find your components and drag them to the Viewer to add them to your app.

Designer Button: Click from any tab to go to the Designer tab.

Properties: Select a Component in the Components List to change its properties (color, size, behavior) here.



Viewer: Drag components from the Palette to the Viewer to see what your app will look like.

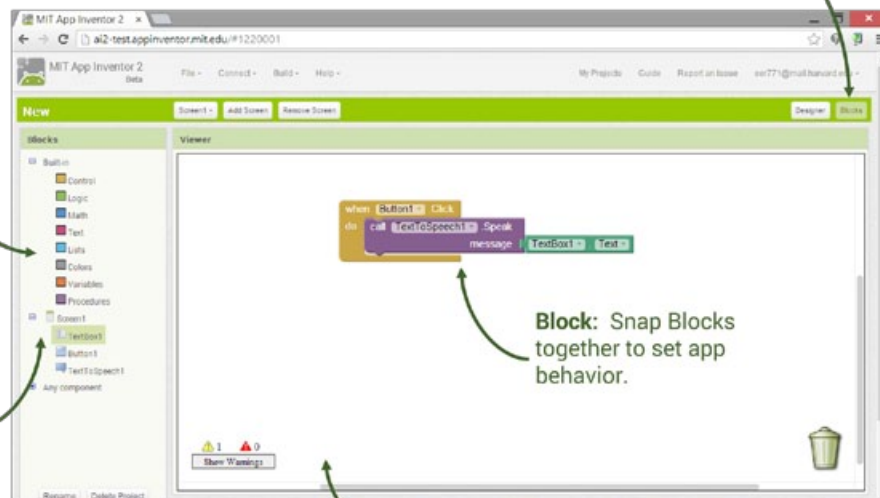
Figure 1.1 The Designer view of the AI2 platform

Built-In Drawers: Find Blocks for general behaviors you may want to add to your app and drag them to the Blocks Viewer.

Blocks Button: Click from any tab to go to the Blocks tab.

Component-Specific Drawers: Find Blocks for behaviors for specific Components and drag them to the Blocks Viewer.

Block: Snap Blocks together to set app behavior.



Viewer: Drag Blocks from the Drawers to the Blocks Viewer to build relationships and behavior.

Figure 1.2 The Blocks view of the AI2 platform

1.4 Device Setup for App Development and Debugging³

You can set up App Inventor and start building apps in minutes. The **Designer and Blocks Editor** run completely in the browser (the cloud). To see your app on a device while you build it (also called **Live Testing**), follow the steps below.

You have three options for setting up live testing while you build apps.

Option 1: If you are using an Android device and you have a wireless Internet connection, you can start building apps without downloading any software to your computer. You will need to install the AppInventor Companion App for your device. This option is **strongly recommended**. (See section 1.4.1.)

Option 2: If you do not have an Android device, you will need to install software on your computer so that you can use the on-screen Android emulator. (See section 1.4.2.)

Option 3: If you do not have a wireless Internet connection, you will need to install software on your computer so that you can connect to your Android device by USB. **The USB Connection option can be tricky, especially on Windows. Use this as a last resort.** (See section 2.4.3.)

³ 1.4 Device Setup for App Development and Debugging is adapted from <http://appinventor.mit.edu/explore/get-started.html> under a Creative Commons Attribution-ShareAlike 3.0 Unported Licence.

System Requirements

COMPUTER AND OPERATING SYSTEM	Macintosh (with Intel processor): Mac OS X 10.5 or higher Windows: Windows XP, Windows Vista, Windows 7
BROWSER	GNU/Linux: Ubuntu 8 or higher, Debian 5 or higher Mozilla Firefox 3.6 or higher (Note: If you are using Firefox with the NoScript extension, you'll need to turn the extension off. See the note on the troubleshooting page.) Apple Safari 5.0 or higher Google Chrome 4.0 or higher IMPORTANT: Microsoft Internet Explorer is not supported
PHONE OR TABLET (OR USE THE ON-SCREEN EMULATOR)	A smartphone with Android Operating System 2.3 ("Gingerbread") or higher





Figure 1.3 Building apps with an Android device and Wi-Fi connection

1.4.1 Building Apps with an Android Device and Wi-Fi Connection

A computer, Android device and Wi-Fi connection are the easiest way to test your apps (Figure 1.3).

With this combination, you can use App Inventor without downloading anything to your computer! You'll develop apps on the AI2 website: <https://ai2.appinventor.mit.edu>. To do live testing on your Android device, simply install the MIT App Inventor Companion app on your Android phone or tablet. Once the Companion is installed, you can open projects in App Inventor on the Web, open the Companion on your device and test your apps as you build them.

The following steps will walk you through the process.

Step 1: Download and install the MIT AI2 Companion App on your phone

Open your device's QR code scanner and scan the QR code (Figure 1.4) to download the **Companion App** from the Google Play Store. If you don't have a QR code scanner installed on your device, you can download one for free from Google Play. **QR Droid Code Scanner** is one example (<https://play.google.com/store/apps/details?id=la.droid.qr&hl=en>).

Alternatively, follow this link from your device: <https://play.google.com/store/apps/details?id=edu.mit.appinventor.aicompanion3>.



Figure 1.4 QR code to download the Companion app from the Google Play Store

After downloading, follow the instructions to install the Companion app on your device. It will then be on your phone or tablet ready for you to use App Inventor.

NOTE: If you cannot use the QR code, use the Web browser on your device to go to the Google Play Store. Look for **MIT AI2 Companion** in the store. When you find Companion, click the **Install** button for the Companion app.



Step 2: Connect both your computer and your device to the SAME Wi-Fi Network

App Inventor will automatically show you the app you are building — but only if your computer (running App Inventor) and your Android device (running the Companion) are connected to the same Wi-Fi network. A more detailed explanation can be found at <http://appinventor.mit.edu/explore/support/explain-wifi-connection.html>.

Step 3: Open an App Inventor project and connect it to your device

Go to App Inventor and open a project (or create a new one: choose **Project** > **Start New Project** and give your project a name). Choose **Connect** and then **AI Companion** from the dropdown menu in the AI2 browser as shown in Figure 1.5.

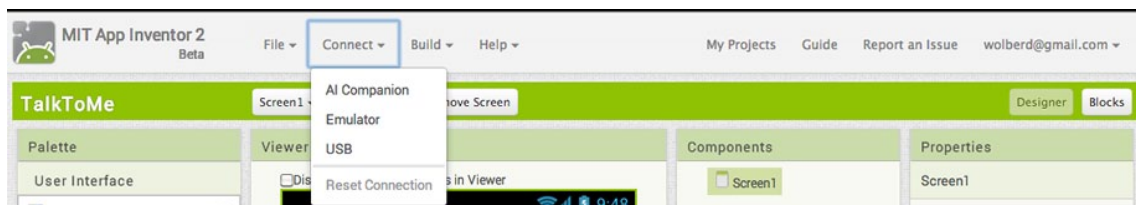


Figure 1.5 Connecting to AI Companion from AI2

A dialog with a QR code will appear on your computer monitor (Figure 1.6). On your device, launch the MIT App Companion app just as you would launch any app. Then click the **Scan QR code** button on the Companion and scan the code in the App Inventor window.

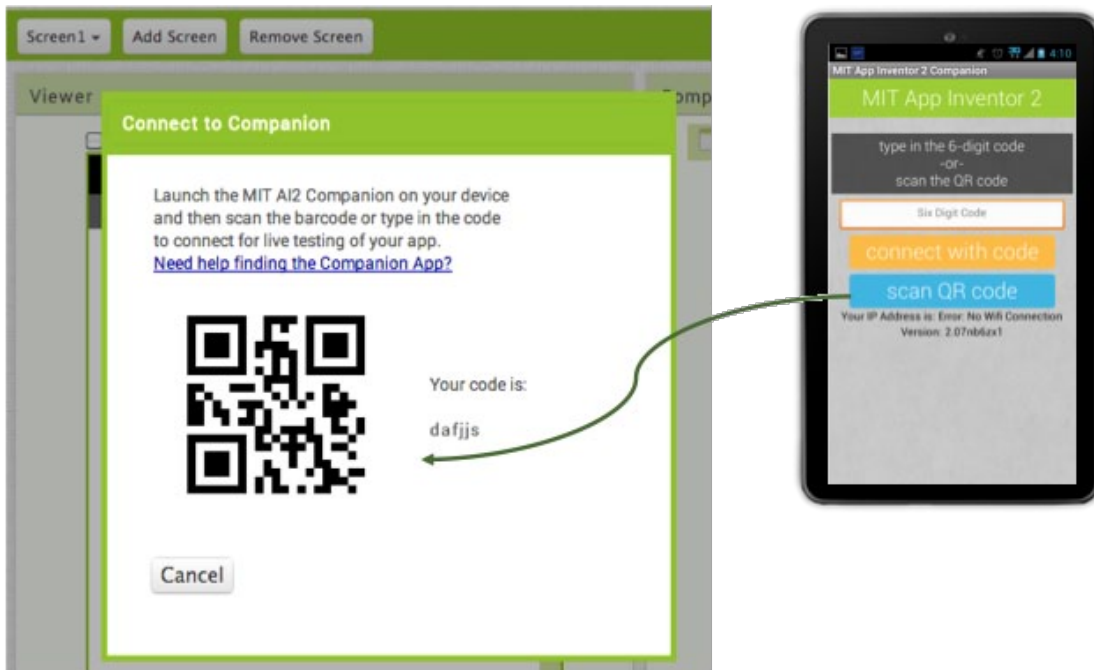


Figure 1.6 Connecting to your project using a QR code or six-character code

Within a few seconds, you should see the app you are building on your device. It will update as you make changes to your design and blocks, a feature called **live testing**.

If you have trouble scanning the QR code or your device does not have a scanner, type the six-character code shown on the computer into the Companion's text area on your Android device exactly as shown. Choose the orange **Connect with code** option.

IMPORTANT: Do not hit **Enter** or make a carriage return. Type only the six characters and then press the orange button.

TROUBLESHOOTING

If your app does not appear on your device, the most likely problems are:

- You may have an outdated version of the App Inventor Companion app. Download the latest Companion app for App Inventor 2 from above.
- Your device may not be connected to Wi-Fi. Make sure you see an IP address at the bottom of the AI Companion app screen on your phone or tablet.
- Your device may not be connected to the same Wi-Fi network as your computer. Make sure both devices are connected to the same Wi-Fi network (check the network name).
- Your school or organisation may have network protocols in place that block the Wi-Fi connection. If this is the case, use App Inventor with the emulator or use a USB cable to connect your device.

1.4.2 Building Apps with the Emulator

If you don't have an Android phone or tablet handy, you can still use App Inventor. Have a class of 30 students? Let them work primarily on emulators (Figure 1.7) and share a few devices.



**Build your project on
your computer** **Test it in real-time on
your computer with
the onscreen
emulator**

Figure 1.7 Using the emulator to build apps

To use the emulator, you first need to install some software on your computer (this is not required for the Wi-Fi option). Follow the instructions below for your operating system.

IMPORTANT:

- If you are updating a previous installation of the App Inventor software, see How to Update the App Inventor Software: <http://appinventor.mit.edu/explore/ai2/update-setup-software.html>.
- You can check whether your computer is running the latest version of the App Inventor software by visiting the App Inventor 2 Connection Test page: <http://appinventor.mit.edu/test>.



Step 1. Install the App Inventor setup software

- Instructions for Mac OS X: <http://appinventor.mit.edu/explore/ai2/mac.html>
- Instructions for Windows: <http://appinventor.mit.edu/explore/ai2/windows.html>
- Instructions for GNU/Linux: <http://appinventor.mit.edu/explore/ai2/linux.html>

Step 2. Launch aiStarter (Windows and GNU/Linux only)

Using the emulator or a USB cable requires the use of a program named aiStarter. This program permits the browser to communicate with the emulator or USB cable.

On a Mac, aiStarter will start automatically when you log in to your account and will run invisibly in the background.

With Windows, there will be shortcuts to aiStarter from your **Desktop**, from the **Start** menu, from **All Programs** or from **Startup Folder**. If you want to use the emulator with App Inventor, you will need to launch aiStarter manually on your computer when you log in. You can start aiStarter by clicking the icon (Figure 1.8) on your Desktop or from your Start menu.

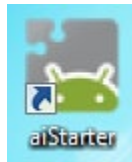


Figure 1.8 The aiStarter icon for Windows

You'll know that you've successfully launched aiStarter when you see a window like the following (Figure 1.9):

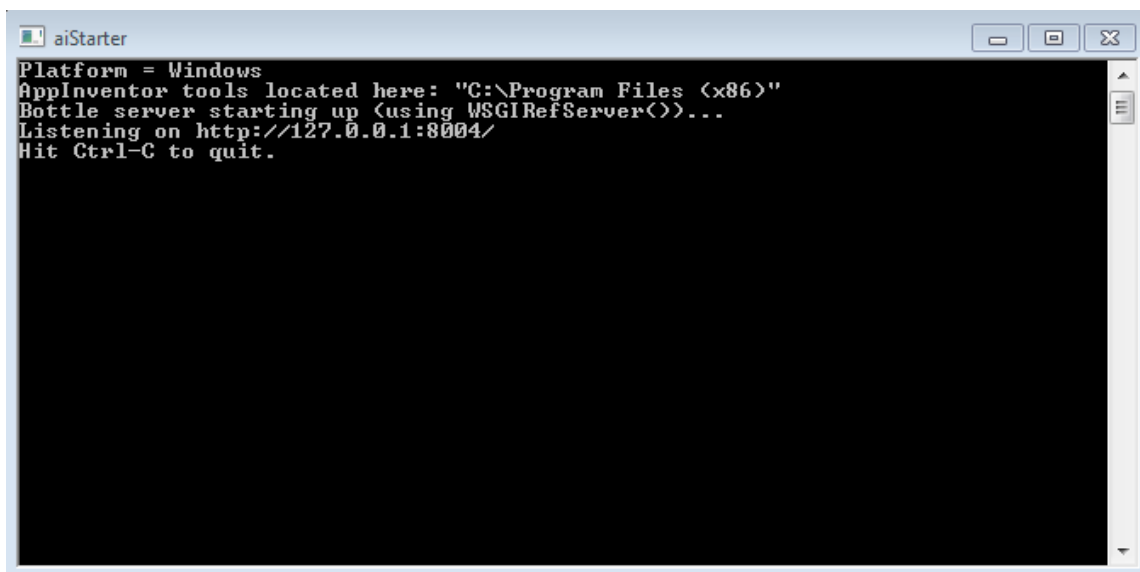


Figure 1.9 aiStarter start-up window

On GNU/Linux, you'll need to launch aiStarter manually from the following folder: **/usr/google/appinventor/commands-for-Appinventor**

You can launch it from the command line with **/usr/google/appinventor/commands-for-appinventor/aiStarter &**

For help with aiStarter, see Connection Help at <http://appinventor.mit.edu/explore/ai2/aistarter-help.html>.

Step 3. Open an App Inventor project and connect it to the emulator

Go to App Inventor and open a project (or create a new one: **Project** > **Start New Project** and give your project a name). Then, from App Inventor's menu (on the App Inventor cloud-based software at <http://ai2.appinventor.mit.edu>), go to the **Connect** menu and click the **Emulator** option as shown in Figure 1.10.

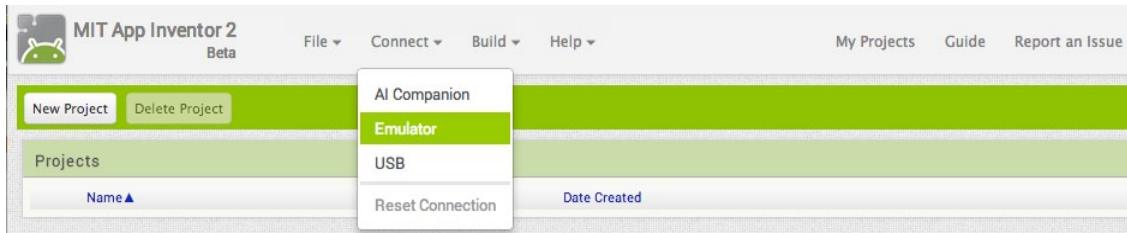


Figure 1.10 Connecting to the emulator from AI2

You'll get a notice saying that the emulator is connecting. Starting the emulator can take a couple of minutes. You may see update screens like Figure 1.11 during the start-up.

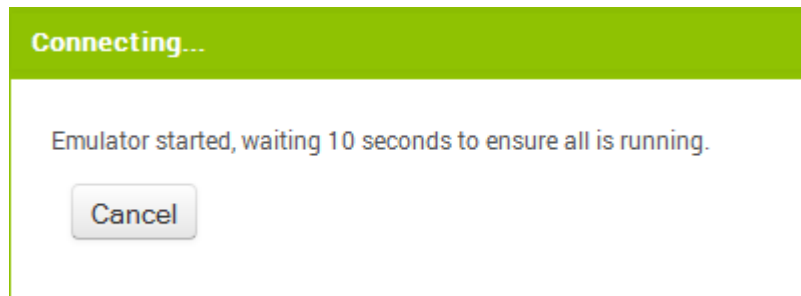


Figure 1.11 Update screens during emulator start-up

The four phases of emulator start-up are shown in Figure 1.12. The emulator will initially appear with an empty black screen (1). The emulator is ready when you see a coloured background (2). Once the background appears, wait until the emulated phone has finished preparing its SD card: there will be a notice at the top of the phone screen while the card is being prepared. Once connected, the emulator will launch and show the app you have open in App Inventor.



Figure 1.12 The four phases of emulator start-up



NOTE: If this is the first time you are using the emulator after installing the App Inventor Setup software, you will see a message asking you to update the emulator. Follow the directions on the screen to perform the update and reconnect the emulator. You will need to do this type of update whenever there is a new version of the App Inventor software.

1.4.3 Building Apps with an Android Device and USB Cable

Some schools' and organisations' firewalls block the type of Wi-Fi connection required. In that case, use a USB cable as shown in Figure 1.13.

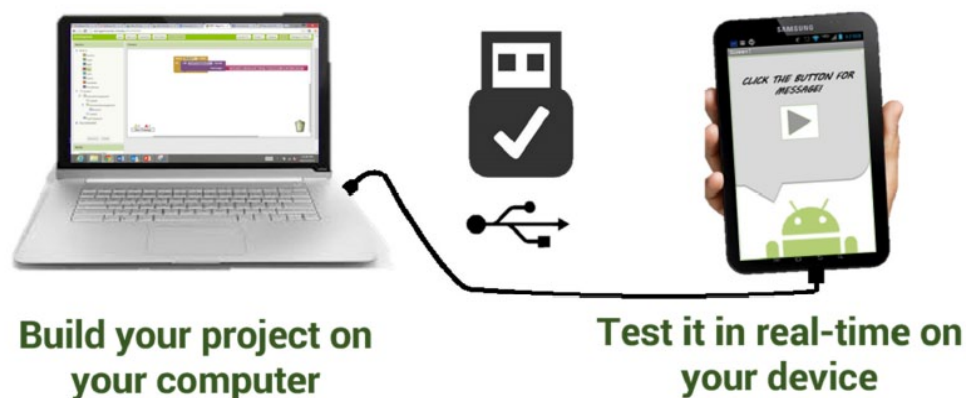


Figure 1.13 Using a USB cable to build apps

When you use App Inventor with a phone or tablet, your device communicates with the App Inventor software running in your computer's browser window. This communication is managed by the AI2 Companion App running on the device. See Step 2 (below) for how to install the AI Companion software.

NOTE: The Companion can communicate with your computer over a wireless connection. **This is the method strongly recommended by the App Inventor team.** It does not require any additional software to be installed on your computer. (See Section 1.4.1.)



There are, however, some environments where wireless connections won't work. For example, some hotels, conference centres and schools configure their wireless networks to prohibit two devices on the network from communicating with each other. See <http://appinventor.mit.edu/explore/support/explain-wifi-connection.html> for a short explanation. Some App Inventor users have solved this problem by purchasing a wireless router and setting up their own local network. (Also, most Macs and some PCs can serve as Wi-Fi routers that can handle a small number of machines.) However, if this is impossible, you can use App Inventor with a phone or tablet if you connect it to the computer with a USB cable.

Setting up a USB connection can be awkward, especially on Windows machines, which need special driver software to connect to Android devices. (This is not the case with Mac or Linux, which do not need special drivers.) Unfortunately, different devices may require different drivers, and, outside of a few standard models, Microsoft and Google have left it to the device manufacturers to create and supply the drivers. Therefore, you may have to search for the appropriate driver for your phone. App Inventor provides a test program that checks if your USB-connected device can communicate with the computer. You should run this test and resolve any connection issues before trying to use App Inventor with a USB cable on a device.

Here are the steps for using App Inventor with a USB cable.

Step 1: Download and install the App Inventor setup software

To connect with the USB, you need to first install the App Inventor setup software on your computer. Follow the instructions below for your operating system.

- Instructions for Mac OS X: <http://appinventor.mit.edu/explore/ai2/mac.html>
- Instructions for Windows: <http://appinventor.mit.edu/explore/ai2/windows.html>
- Instructions for GNU/Linux: <http://appinventor.mit.edu/explore/ai2/linux.html>

IMPORTANT:

- If you are updating a previous installation of the App Inventor software, see How to Update the App Inventor Software at <http://appinventor.mit.edu/explore/ai2/update-setup-software.html>.
- You can check whether your computer is running the latest version of the software by visiting the Connection Test Page: <http://appinventor.mit.edu/test>.



Step 2: Download and install the MIT AI2 Companion App on your phone

Open your device's QR code scanner and scan the QR code (Figure 1.14) to download the Companion App from the Google Play Store. If you don't have a QR code scanner installed on your device, you can download one for free from Google Play. **QR Droid Code Scanner** is one example (<https://play.google.com/store/apps/details?id=la.droid.qr&hl=en>).

Alternatively, follow this link from your device: <https://play.google.com/store/apps/details?id=edu.mit.appinventor.aicompanion3>.



Figure 1.14 QR code to download the Companion app from the Google Play Store

After downloading, follow the instructions to install the Companion app on your device. It will then be on your phone or tablet ready for you to use App Inventor.



NOTE: If you cannot use the QR code, use the Web browser on your device to go to the Google Play Store. Look for **MIT AI2 Companion** in the store. When you find Companion, click the **Install** button for the Companion app.

Step 3. Launch aiStarter (Windows & GNU/Linux only)

Using the emulator or a USB cable requires the use of a program named aiStarter. This program permits the browser to communicate with the emulator or USB cable.

On a Mac, aiStarter will start automatically when you log in to your account and will run invisibly in the background.

With Windows, there will be shortcuts to aiStarter from your **Desktop**, from the **Start** menu, from **All Programs** or from **Startup Folder**. If you want to use the emulator with App Inventor, you will need to manually launch aiStarter on your computer when you log in. You can start aiStarter by clicking the icon (Figure 1.15) on your Desktop or from your Start menu.

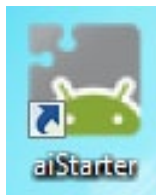


Figure 1.15 The aiStarter icon for Windows

To launch aiStarter using Windows, double-click on the icon. You'll know that you've successfully launched aiStarter when you see a window like the following (Figure 1.16):

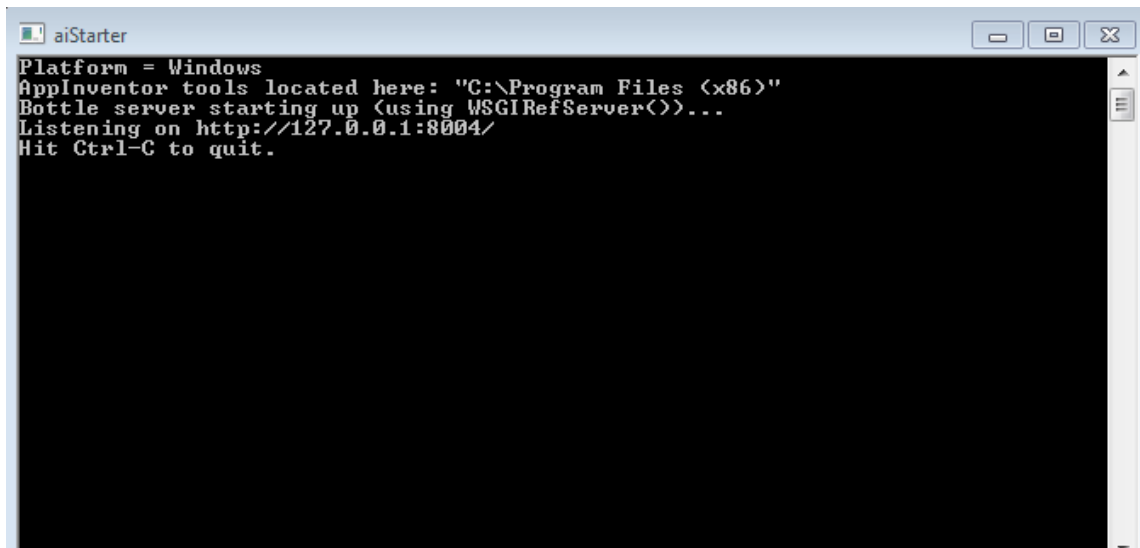


Figure 1.16 aiStarter start-up window

On GNU/Linux, you'll need to launch aiStarter manually from the following folder: **/usr/google/appinventor/commands-for-Appinventor**

You can launch it from the command line with **/usr/google/appinventor/commands-for-appinventor/aiStarter &**

For help with aiStarter, see <http://appinventor.mit.edu/explore/ai2/aistarter-help.html>.

Step 4: Set up your device for USB (turn USB Debugging on)

On your Android device, go to **System settings** then **Developer options**. Turn on **Developer options** and be sure that **USB debugging** is allowed.

On most devices running Android 3.2 or older, you can find this option under **Settings > Applications > Development**.

On Android 4.0 and newer, it's hidden by default. To make it available, go to **System settings > About phone > Software information** and tap **Build number** seven times. Return to the previous screen to find **Developer** options, including **USB Debugging**.

Step 5: Connect your computer and device, and authenticate if necessary

Connect your Android device to the computer using a USB cable. Be sure that the device connects as a **mass storage device** (not **media device**) and that it is not mounted as a drive on your computer. You may have to go to the **Finder** (on a Mac) or **My Computer** (on Windows) and disconnect any drive(s) that were mounted when you connected your Android device.



NOTE: On Android 4.2.2 and newer, the first time you connect your device to a new computer the message “Allow USB Debugging?” will pop up. Press **OK**. This authenticates the computer to the device, allowing the computer to communicate with it. You’ll need to do this for each computer you want to connect to the device, but only once per computer.

Step 6: Test the connection

Go to the Connection Test page at <http://appinventor.mit.edu/test> to see if you get a confirmation that your computer can detect the device.



TROUBLESHOOTING: If the test fails, go to General Connection Help at <http://appinventor.mit.edu/explore/ai2/connection-help.html> and choose the **USB help** option for your computer (Windows or Mac). You won’t be able to use App Inventor with the USB cable until you resolve the connection issues.

2 Activities

You are now ready to start building Android apps on AI2. The Activities and Tutorials are designed to give you hands-on training in most of the basic components available on AI2 for app development. The Activities are smaller apps that will have only one or two functions. The Tutorials are considerably more sophisticated apps with several interrelated functions. You should complete all the Activities before attempting the Tutorials.

Walkthrough:

Start a new project for each Activity/Tutorial. Name the project accordingly. Click on **Screen 1** on the **Designer** view to change the **Properties** of the app such as Title and Description. Drag and drop the **Components** from the **Palette** onto the **Viewer** to build the user experience (UX) of your application (Figure 2.1). If you have configured Live Testing, you should now be able to see your app take shape on your Android device or emulator.

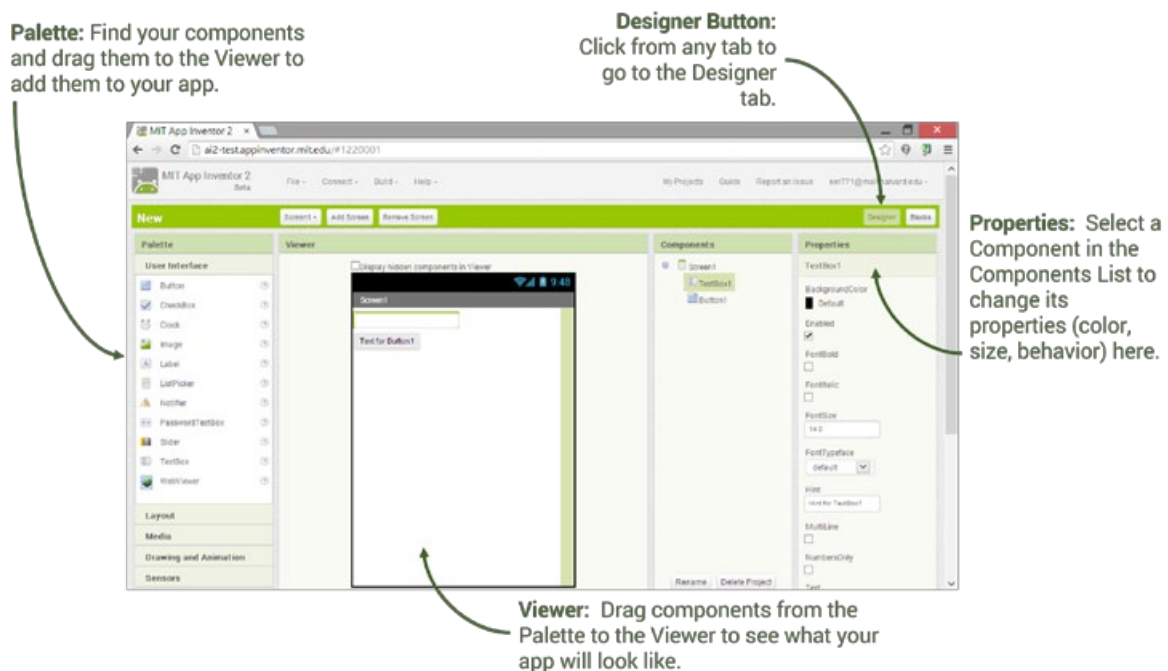


Figure 2.1 Using the Palette, Viewer and Properties in Designer view

Once you have designed the look and feel of your app on the **Viewer**, you need to build the logic of the application. This is done using the **Blocks** view. Click on the **Blocks** button to access the Blocks view (Figure 2.2).

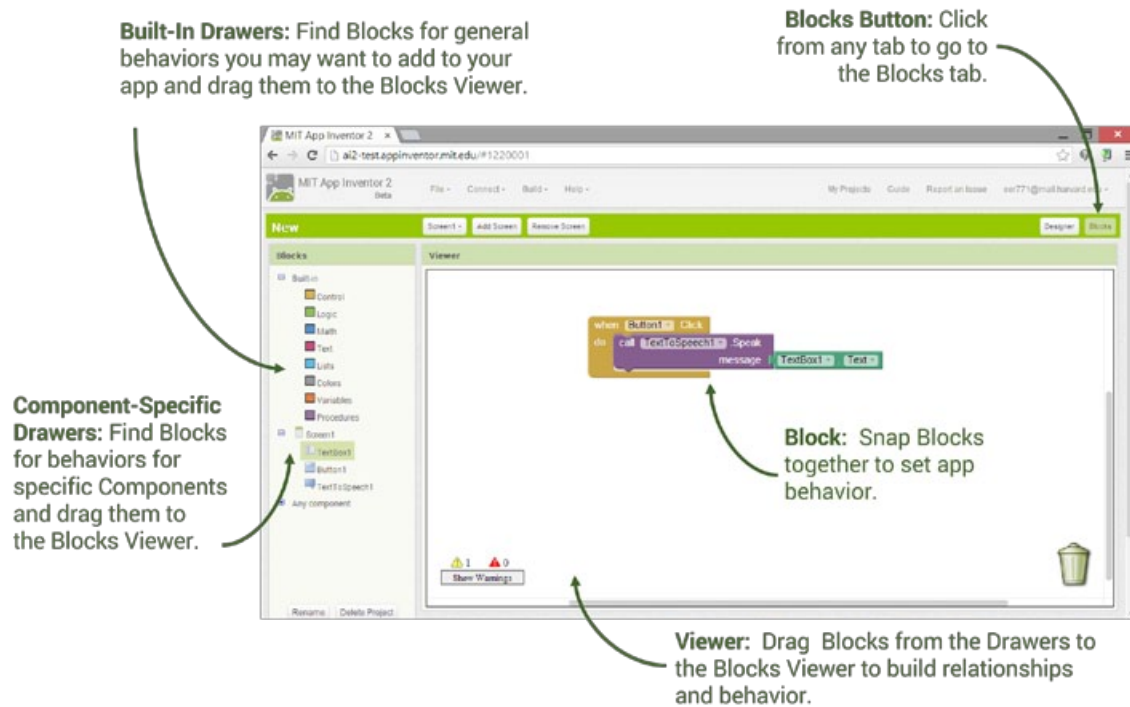





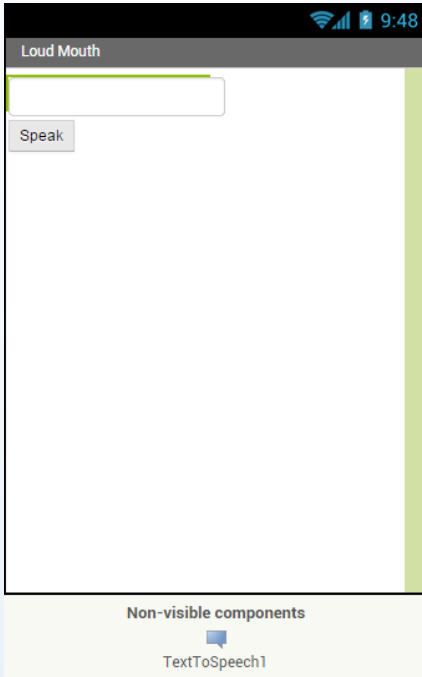




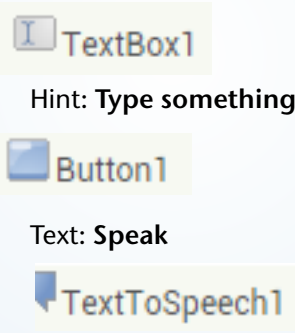
Figure 2.2 Using the Blocks drawers in Blocks view

Find the blocks needed to build your application from the **Built-In Drawers** and the **Component-Specific Drawers**. Drag and drop the blocks onto the empty space in the **Viewer** to build your logic. If the blocks fit each other, they will snap together. To delete a block, press **DEL** or drag and drop the block into the trash can.

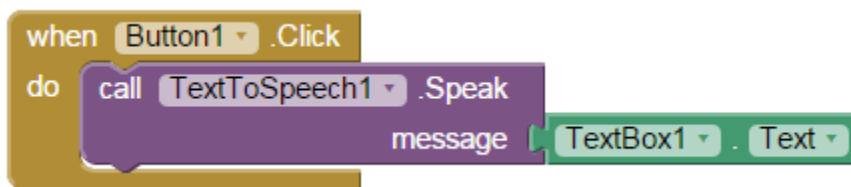
You can download the resources — images, audio files, etc. — for the activities and tutorials from the Resources link in each Activity/Tutorial sheet. All Resources for the Activities and Tutorials are available at <https://goo.gl/Abz5BX>.

Activity 1: TextToSpeech (Loud Mouth)

 Project name (AppName)	COL_LoudMouth
 App Name (Title)	Loud Mouth
 App description (AboutScreen)	This app speaks out the text typed into the TextBox using the TextToSpeech component.
 Resources	https://goo.gl/8HdAIJ
 Screenshot	

 Expected Learning Outcomes	<p>By the end of this activity you should be able to:</p> <ul style="list-style-type: none"> • Add components into the Designer • Use the Blocks editor • Add a TextBox and Button to the interface • Use the Non-visible TextToSpeech component
 Components & Attributes	 <p>Hint: Type something</p> <p>Text: Speak</p>

Blocks



Walkthrough:

1. Log in to your AI2 account at <http://ai2.appinventor.mit.edu>.
2. Click on the **Start new project** button (Figure 2.3) to begin a new project.

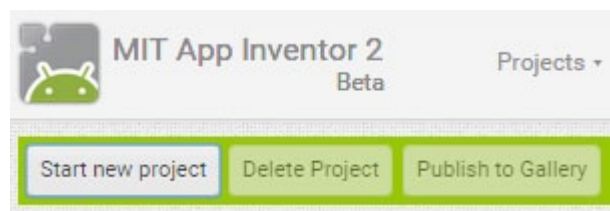


Figure 2.3 Click on the Start new project button to open a new AI2 project

3. Name the project according to the project name (Figure 2.4) given in the Activity sheet. In this case, the project name is COL_LoudMouth. You can name your project any way you like as long as it contains only alphanumeric characters and underscores. You cannot include spaces in your project name (although you can use them in the app name). It is also good programming practice to use **CamelCase notation** as your naming convention. This is the practice of using uppercase letters to indicate the beginning of each new word. More details on CamelCase notation can be found at <https://en.wikipedia.org/wiki/CamelCase>.

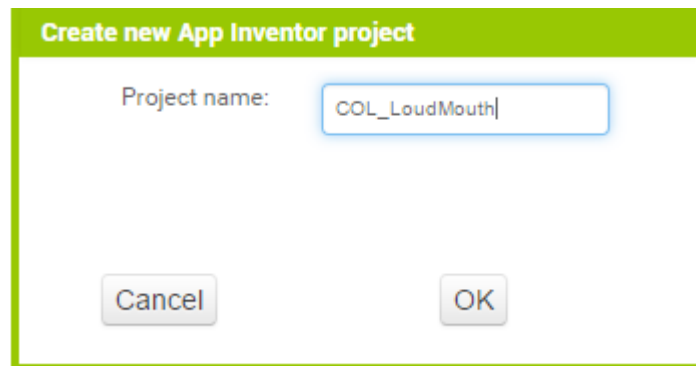


Figure 2.4 Name your project using the project name provided in the activity sheet

4. Once you have created the new project, you will enter the **Designer** view inside AI2. The project name will be displayed as shown in Figure 2.5.

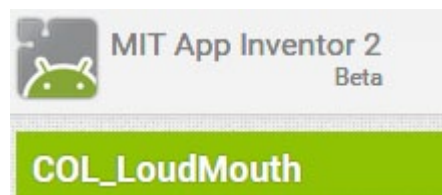


Figure 2.5 The project name is displayed inside the AI2 project

5. A blank phone screen will be visible within the **Designer** view as shown in Figure 2.6. This represents what the user sees on their Android mobile phone/tablet screen. You will be developing the visual aspects of your app on this blank screen. Notice that the screen indicates the Wi-Fi status, signal and battery strength, and time at the top just like the screen of an Android mobile device.

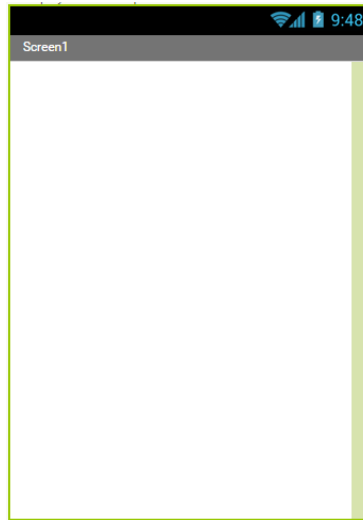


Figure 2.6 Blank phone screen used to develop the visual aspects of the app

6. It is good practice to start your Live Testing at this point so that you are aware of how your app is taking shape on the actual device. Since you already know how to set up and start Live Testing from Section 1, you can select which type of testing you want to do (**AI Companion**, **Emulator** or **USB**) from the **Connect** menu. Once you have connected to your phone or the emulator, you will see the empty phone screen shown in Figure 2.6 on your device.
7. Note that Figure 2.6 displays the words “Screen 1” on the top of the phone screen. This is usually the **app title**. The AI2 platform uses the title Screen 1 by default. Change it to the **app name** given in the Activity sheet. In this case, Loud Mouth. To do this, click on the grey strip that displays the title then navigate to **Properties** on the far right. Here you will first see an empty text box called **AboutScreen** at the very top of the pane. This is the information the user will see when the app is being installed on their device or when they check the app in the installed apps list. You will need to provide a meaningful description for your app in this section. For the purposes of this toolkit, use the **app description** in the Activity sheet (Figure 2.7). Scroll down the **Properties** pane to locate the **Title** text box and change the title from Screen 1 to Loud Mouth. You will now see that the title on the phone screen has changed to Loud Mouth as shown in Figure 2.8.

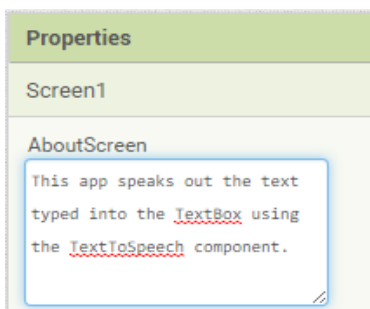


Figure 2.7 Provide a meaningful description of the app

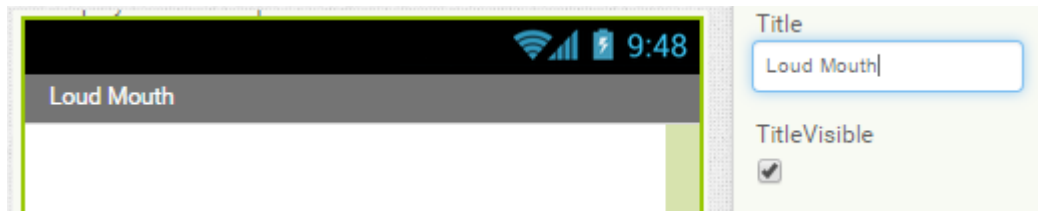


Figure 2.8 Change the title of the app

8. Now you are ready to build your first app. To do this, you will need to use three components as shown in the **Components and Attributes** section of the Activity sheet. Navigate to the **Palette** located on the left of the screen and look for the first two components, **TextBox** and **Button**, as shown on Figure 2.9.

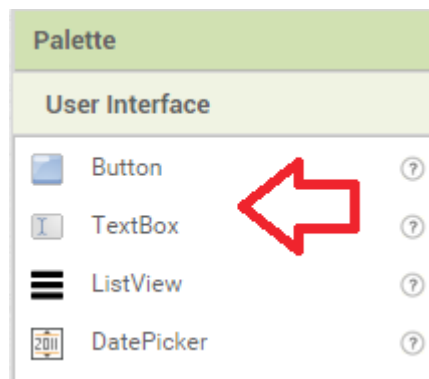


Figure 2.9 Locate the TextBox and Button components

9. Drag and drop the **TextBox** followed by the **Button** onto the blank phone screen as shown in Figure 2.10. You will immediately see these components on your Live Test device.

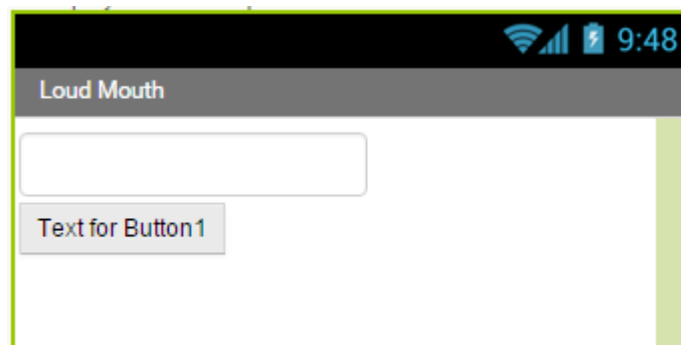


Figure 2.10 Drag and drop TextBox and Button onto the blank phone screen

10. Now locate the **TextToSpeech** component located inside the **Media drawer** within the **Palette** as shown in Figure 2.11. Drag and drop it onto the phone screen as you did with the TextBox and Button components. Note that the TextToSpeech component does not appear on the phone screen. This is because it is classified as a **non-visible component**. These components are invisible to the user. They do a certain task in the background. You will see these non-visible components at the bottom of the phone screen outside the visible area as shown in Figure 2.12.

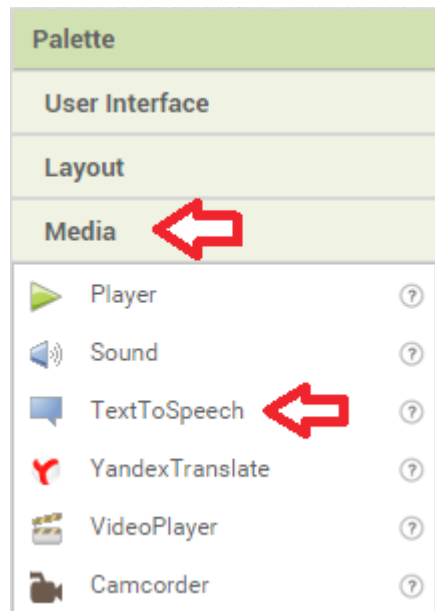


Figure 2.11 TextToSpeech component located within the Media drawer of the Palette

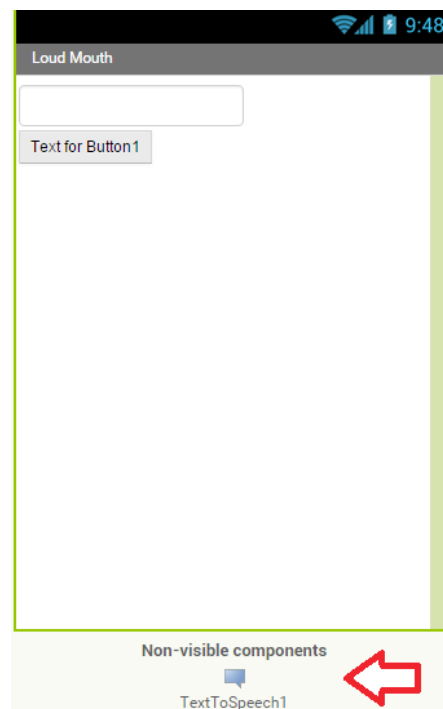


Figure 2.12 Non-visible components shown outside the visible area of the phone screen

11. Now that you have all the components on the screen, it's time to change their **Properties** according to the Components and Attributes section of the Activity sheet. The words “properties” and “attributes” mean the same thing. First click on the **TextBox** on the phone screen. This will highlight the component. Now navigate to the **Properties** section as you did in Step 7. Scroll down to find the property **Hint** and change it to “Type something...” as shown in Figure 2.13. This property gives the user a hint about what to do with the textbox inside the app. You will now see the words “Type something...” in light grey within the TextBox of your Live Test device. When you click on the TextBox on the Live Text device this message will disappear, allowing the user to type some text on the virtual keyboard. Note that the Hint is only a guide to the user. It is not the contents of the TextBox. The user has to type something into the TextBox for the app to work.

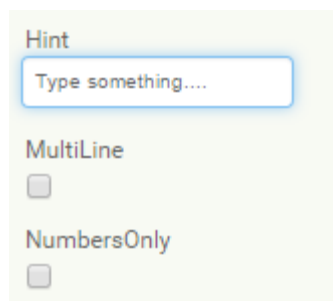


Figure 2.13 Change the Hint property of the TextBox component

12. Now click on the **Button** component on the phone screen in **Designer** view to change its properties. Locate the property **Text** and change it to “Speak.” You will see that the text on the button face has changed from “Text for Button 1” to “Speak” (Figure 2.14).



Figure 2.14 Change the Text property of the button to “Speak”

13. No property changes are mentioned in the Activity sheet for the TextToSpeech component, so leave the default properties as they are.
14. Congratulations! You have finished building the visual design of your first app. You will be able to see it on your Live Test device. However, nothing happens when you press the **Speak** button, because you need to build the logic that drives the app. This is done using the **Blocks** view. You can change between the **Designer** and **Blocks** views using the two buttons located on the far top right (Figure 2.15).



Figure 2.15 Use the Designer and Blocks buttons to change between the Designer view and the Blocks view

15. For the app to work, you need to build the logic block given under the Blocks section of the Activity sheet. In this case, the block is as shown in Figure 2.16.

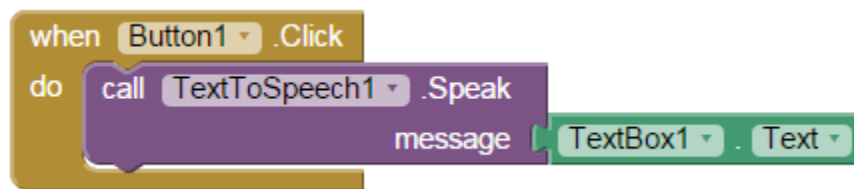


Figure 2.16 The logic block for the LoudMouth app

16. The logic blocks in AI2 are extremely high level in terms of programming. That means that the programming is done using logic that can be built using our own human language (not machine language). You will need to get into the habit of building sentences of what you want the app to do before building the logic blocks. Imagine that you are explaining to a user what the app does. In this case, your explanation will be something like:

The user will enter some text into the text box. When the Speak button is clicked, the app will speak out the text the user has entered.

Considering the above, the app will not have to do anything with respect to the user inputting some text into the TextBox. That is an activity under the user's control. The app only needs to act when the Speak button is pressed. Then the app needs to get the text from the TextBox, convert it to speech and play it back through the speaker. This is the part we need to build with the logic blocks.

Now let's make it a bit more specific with component names. Note that AI2 assigns a number to the default name of each component. The first component will be named XX1. For example, the first button dragged and dropped into the screen will be Button1 and the second will be Button2. The first text box will be TextBox1, and so forth. You can change the names of these components so that they make more sense when building logic blocks. We will look at that in a later Activity. For now, let's stick to the default component names.

The app needs to do the following:

When Button1 is Clicked, TextToSpeech1 reads the Text inside TextBox1 as the Message and Speaks it out.

Pay special attention to the nouns and verbs underlined in the above sentence. These are what we need to build within our logic block.

- Find **Button1** from the **Blocks** section on the left of the **Blocks** view. Click on **Button1** to open the blocks drawer for the **Button1** component. Locate the **When Button1.Click do** block. Drag and drop it into the blank space in the Blocks view as shown in Figure 2.17. Note that blocks are non-visible components and will not show on the Live Test phone screen. However, you will be able to use the functionality on the Live Test device as soon as a complete block is built.

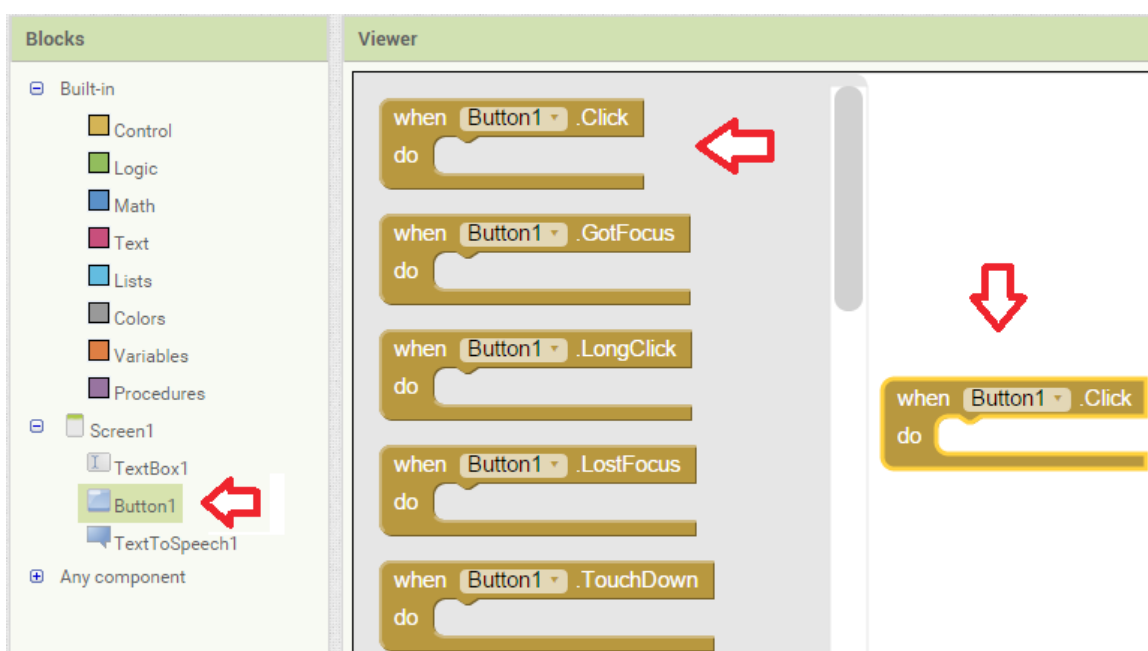


Figure 2.17 Click on *Button1* in the *Blocks* section and drag and drop the *when Button1.Click do* block into the empty space.

- Similarly, click on **TextToSpeech1** to locate the **call TextToSpeech1.Speak** block. Drag and drop it into the groove in the **when Button1.Click do** block as shown in Figure 2.18. You should hear a clicking sound as the two blocks snap together.

NOTE: Only compatible blocks will snap together in this way. If two blocks don't snap together, your logic is faulty.



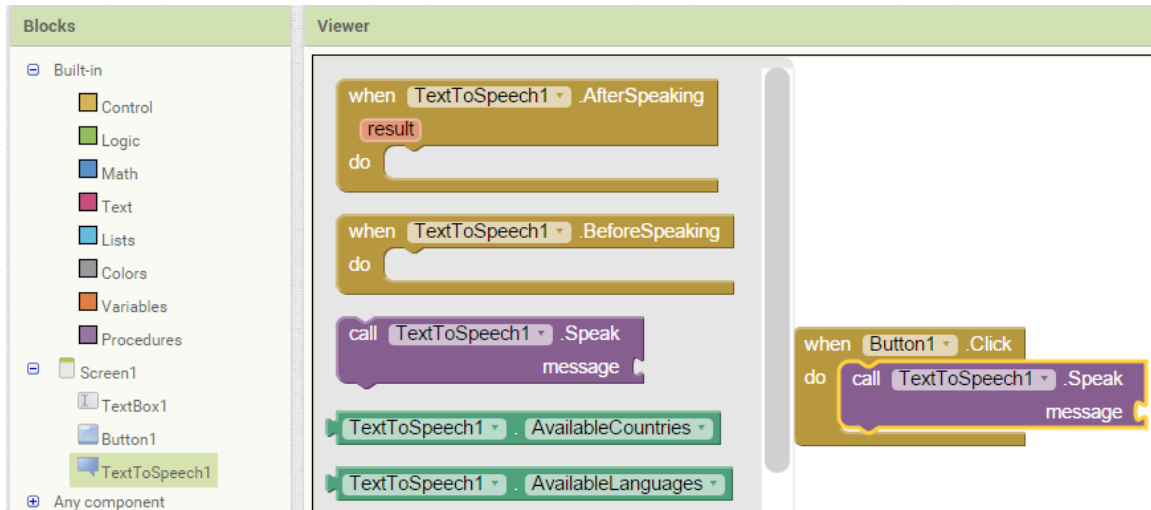


Figure 2.18 Join the two blocks so they click together

19. Finally, click on **TextBox1** and drag and drop the **TextBox1.Text** block so that it fits the groove in the **call TextToSpeech1.Speak message** block and they snap together as shown in Figure 2.19.

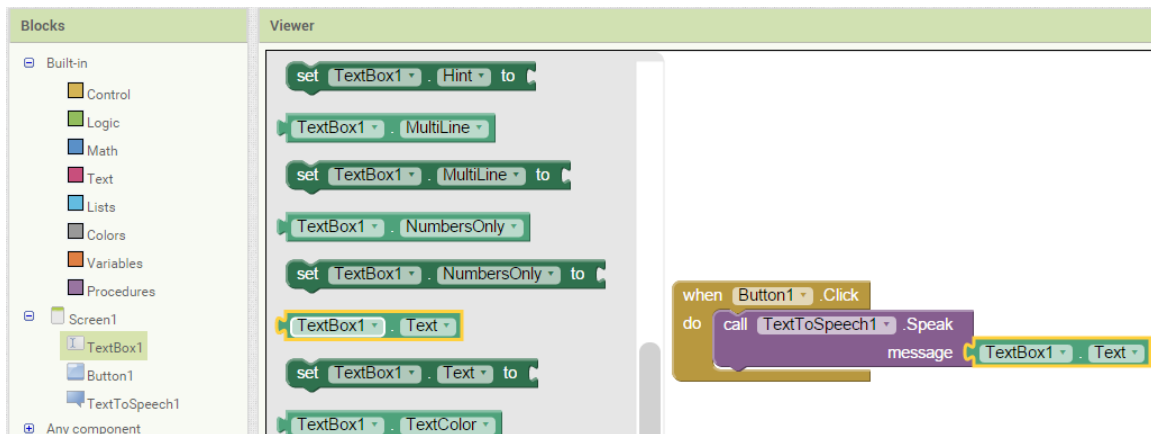










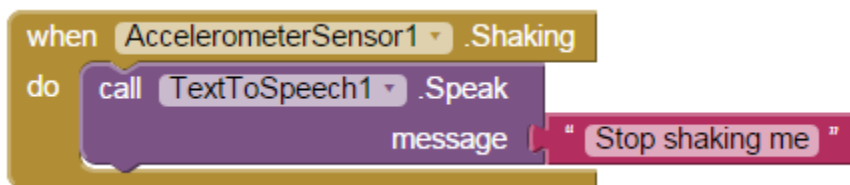
Figure 2.19 Join all three blocks to make the logic of your app

20. Congratulations! You have just built your first Android app with zero programming. You can now test the app on your Live Test device. Type the text “Hello World. I just built my first Android app” into the TextBox and hit the **Speak** button to see what happens.

Activity 2: AccelerometerSensor (Stop Shaking Me)

 Project name (AppName)	COL_StopShakingMe
 App Name (Title)	Stop Shaking Me
 App description (AboutScreen)	This app uses the AccelerometerSensor to detect if the phone is moving or shaking. When you shake the phone it will ask you to stop shaking it.
 Resources	https://goo.gl/85u95x
 Expected Learning Outcomes	By the end of this activity you should be able to: <ul style="list-style-type: none"> • Use the AccelerometerSensor to detect phone movement
 Components & Attributes	 AccelerometerSensor1 <ul style="list-style-type: none"> • MinimumInterval: 1000  TextToSpeech1 <ul style="list-style-type: none"> • Pitch: 2.0

Blocks



Walkthrough:

1. Having developed your first Android app, to convert text to speech, you must be eager to develop your second one. This app is not too different from the **Loud Mouth** application with respect to the functionality and setup. However, one major difference is that there are no visual components in this app — that is, all the components are non-visible components. Therefore, the user will only see a blank white screen on their Android device. You might think that an app needs some form of visual component. However, the modern concepts of app design tend to concentrate on your other senses as well as sight. For example, much emphasis is put on the auditory and tactile components of the user experience (UX) in games. This is of special interest to mobile device users who have some form of disability. You as a developer need to ensure that your app is accessible to as many users as possible.
2. You are now familiar with the process of locating components from the **Palette**. Drag and drop the **AccelerometerSensor** component, which is in the **Sensors drawer** of the **Palette**, onto the phone screen. You already know where to find the **TextToSpeech** component from the previous activity.
3. Change the **MinimumInterval** property of the **AccelerometerSensor** to 1000. Note that time-related properties in AI2 are set in milliseconds — that is, when you set the property to 1000, the AccelerometerSensor checks the movement of the phone every second.
4. Set the **Pitch** property of the **TextToSpeech** component to 2.0. You can set a value between 0 and 2. Lower values give a lower pitch to the voice while higher values give it a higher pitch. Note that you can find out more about each component and its properties by putting your cursor on the **question mark (?)** located next to each component in the **Palette**. A popup (Figure 2.20) will not only give you an indication of what the component does but will also give you a link to more information at the AI2 website, which lists helpful details about all the properties you can set.

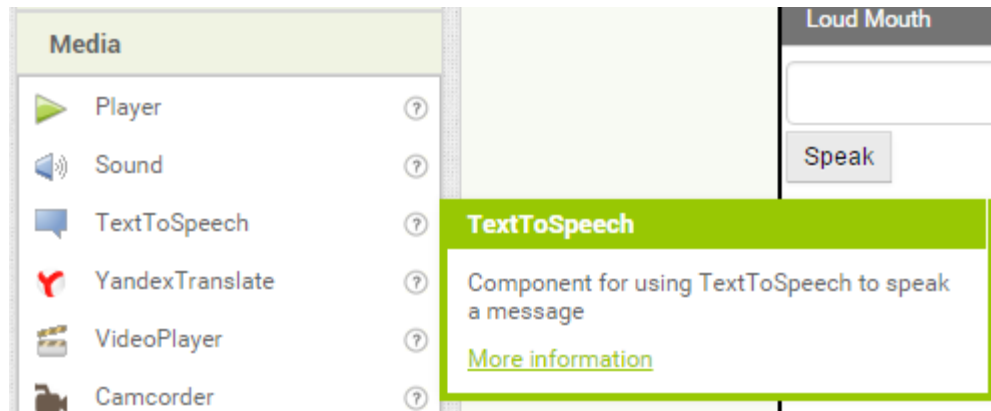


Figure 2.20 More information popup for the TextToSpeech component

- It is very important that you continue to practise the skill of explaining the functionality of an app using simple English. This helps in building logic blocks that are programmatically sound. Let's try to explain the functionality of this app to a user:

When you shake the phone it will speak to you and ask you to stop shaking it.

Now let's put it in a more logical manner:

When **AccelerometerSensor1.Shaking** do XYZ

NOTE: Many programming languages use the dot operator. It means words like “is” and “are” (which are called **stop words**) are replaced with a more machine-friendly format separated by a DOT “.” Furthermore, the **do** is a more definitive way of indicating what action is taken as a result. For example, “I eat when I’m hungry” can be reworded more precisely as “When I’m Hungry do Eat.”



Now, in the context of this app, XYZ means “**Speak out the words ‘Stop shaking me.’**” Therefore, the logic can be written as follows:

When **AccelerometerSensor1.Shaking** do (Speak out the words “Stop shaking me”)

This translates to the block in the Activity sheet as shown in Figure 2.21.

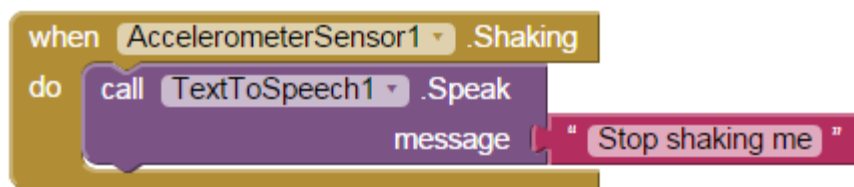


Figure 2.21 The logic block for the Stop Shaking Me app

Since this app is very similar to the Loud Mouth app, I'm sure you can assemble the **[when AccelerometerSensor1.Shaking]** block and the **[call TextToSpeech1.Speak]** blocks. The only difference is that you will be replacing the user input read through a TextBox with a static piece of text. To do this you need to use a block from the **Built-in blocks**, which are predefined blocks. Click on the **Text** and select the first block, which is an **empty Text block**, and join it with the **[call TextToSpeech1.Speak]** block as shown in Figure 2.22.

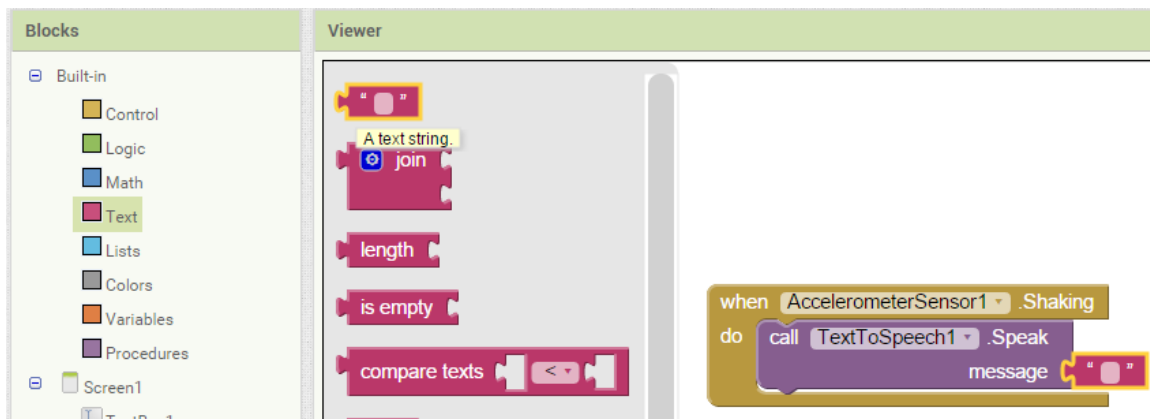





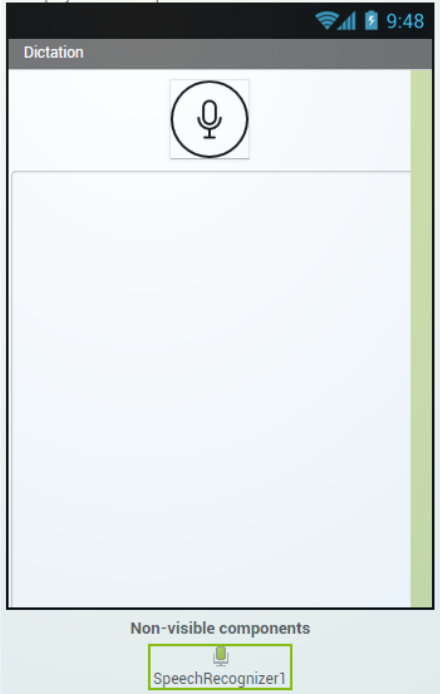


Figure 2.22 Using an empty Text block

6. Now you have to fill in some text in the empty **Text** block. Just click on the empty box in the middle and type “Stop shaking me.”
7. Congratulations! You just finished your second app. Give the phone a shake and see what happens. You can modify the **Pitch** of the **TextToSpeech** component to go from a male voice to a female voice. I hope you are using an actual Android device for your Live Testing because it is almost impossible to shake an emulator.

Activity 3: SpeechRecognizer (Dictation)

 Project name (AppName)	COL_Dictation
 App Name (Title)	Dictation
 App description (AboutScreen)	This app uses the SpeechRecognizer component to take dictation from you.
 Resources	https://goo.gl/XYCiqM
 Screenshot	



Expected Learning Outcomes

By the end of this activity you should be able to:

- Use the SpeechToText component
- Apply an image to a button
- Manipulate Height, Width and Align attributes
- Concatenate text strings using Join



Components & Attributes

Screen1

- AlignHorizontal: **Center**

Button1

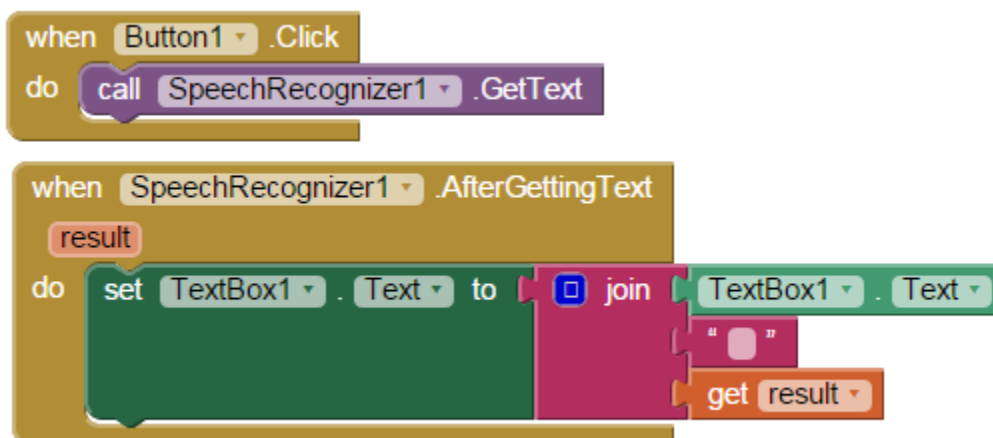
- Text:
- Image: **microphone73.png**

TextBox1

- Hint: **Your dictation appears here...**
- Width: **Fill parent**
- Height: **Fill parent**

SpeechRecognizer1

Blocks



Walkthrough:

1. You learned how to change the **Title** of the **Screen1** component in the Loud Mouth app. However, the component name remains Screen1. You might now wonder if a single app could have multiple screens because they are named 1, 2, 3... Yes, a single app can have many screens, but that is beyond the scope of this toolkit. You can develop extremely powerful apps with just one screen — as you will realise once you start the Tutorials.

Just as you changed the **Title** and **AboutScreen** properties of **Screen1**, set the **AlignHorizontal** property to **Center**, which will ensure that all components on the screen will horizontally align to the centre of the screen, unlike the Loud Mouth app where the components were aligned to the left by default. The dropdown alignment menu gives you the options of **left**, **center** and **right**. The properties for the **AlignVertical** property are **top**, **center** and **bottom**.

2. You learned how to change the text of a **Button** component. Now you will learn how to change the **Image** attribute, which is what you will use for almost all of your future apps. The Image attribute sets an image file such as a .jpeg or .png as the background of a Button. In this case it uses the file **microphone73.png** which is available in the **Resources** pack (see the Resources section of the Activity sheet). Once you have downloaded the Resources pack, extract it to a convenient location. Then use the **Image** property of **Button1** to upload the Image onto the AI2 server as shown in Figure 2.23.

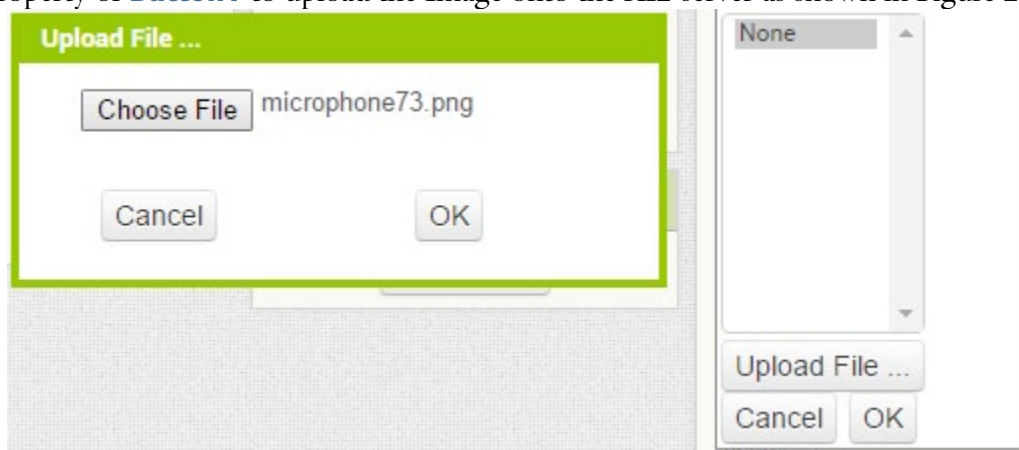


Figure 2.23 Upload an image from the Resources pack

3. **Button1** will now display the image of a microphone. However, you will still have “Text for Button1” in the middle. To remove this, just keep the **Text** property empty.
4. You will find the **SpeechRecognizer** component inside the **Media drawer** of the **Palette**. This is an app with two independent logic blocks. Build the first logic (Figure 2.24) block as follows:

When **Button1.Click**, capture the speech of the user until the user stops speaking.



Figure 2.24 The first logic block, which captures the user's speech

5. Since this is a dictation app, the app is supposed to record a long speech sentence by sentence, just like a secretary would do. Therefore, any new sentence spoken into the app needs to be appended to the previous sentences. So the logic needs to be:

When the user has stopped speaking, append the new text to the existing.

This is done using the block shown in Figure 2.25.

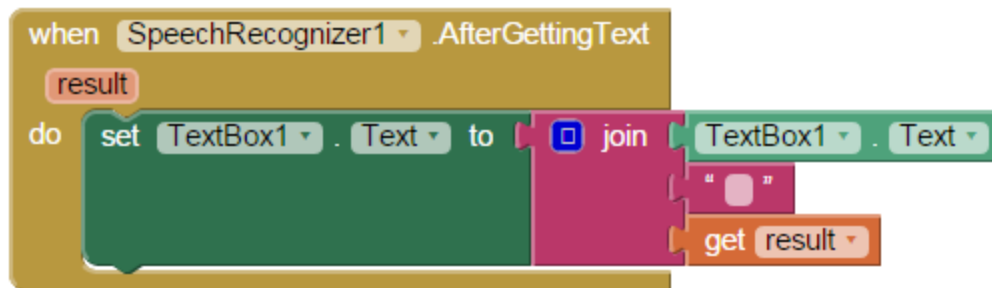


Figure 2.25 Appending text to the existing text when the user has stopped speaking

6. Two aspects will be new to you. The first is how to increase the number of grooves in the **join** block located within the **Text drawer** of the **Built-in blocks**. Click on the blue square. This will give you the option of adding or deleting as many grooves as you want as shown in Figure 2.26. To add a groove, just drag and drop a **string** block into the join block. To remove a groove just select one and press **DEL**.

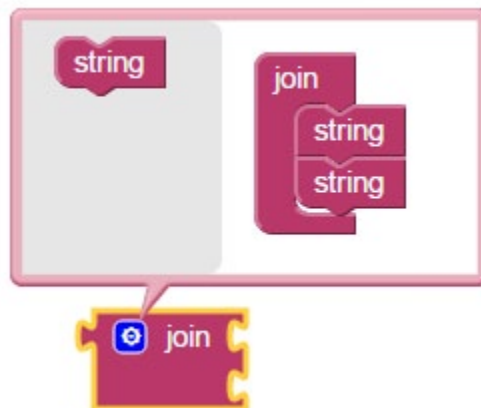


Figure 2.26 Adding and deleting grooves in a join block

7. The second aspect would be how to get the **[get result]** block from the **[when SpeechRecognizer.AfterGettingText do]** block. Just put your cursor on (but **do not click**) the **Result** parameter of the **[when SpeechRecognizer.AfterGettingText do]** block. This will result in a popup as shown in Figure 2.27. Just drag out the **[get result]** block and use it. This block holds the text that was spoken by the user. The Result parameter is referred to as the **return value** of the **[SpeechRecognizer.AfterGettingText]** block. A single block could have multiple return values.

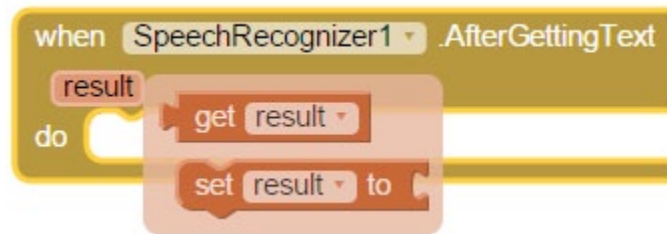





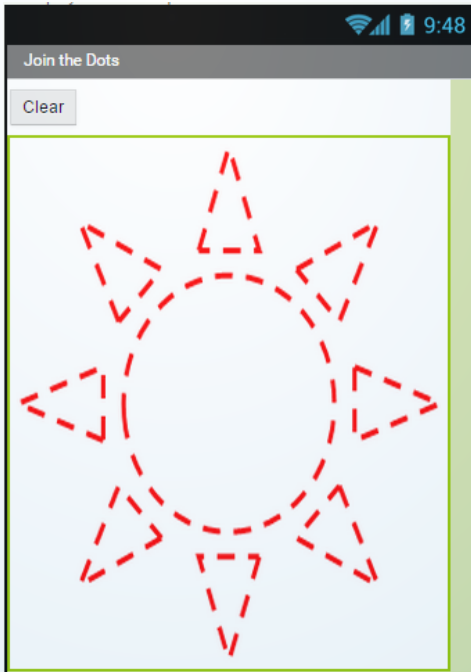
Figure 2.27 Getting the result in a SpeechRecognizer.AfterGettingText block




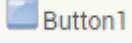
8. Congratulations! That's app number 3. Give it a go on your Live Test device. You should be able to dictate your speech while the app converts it to text which can be emailed or texted as you wish. If you are using an emulator, make sure you have a microphone connected to your PC.

In the Resources:

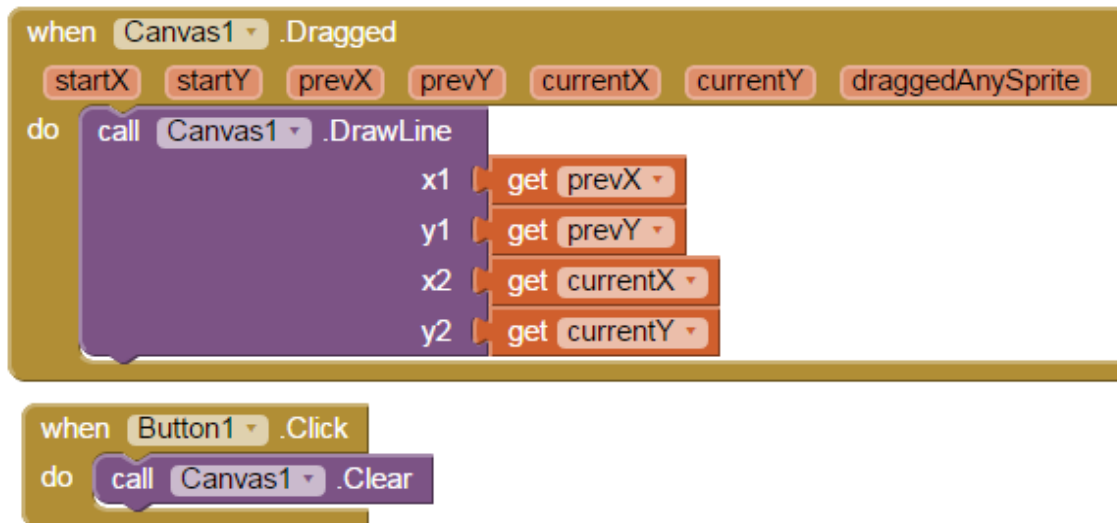
“microphone73” by Sebastien Gabriel <http://sebastien-gabriel.com> from www.flaticon.com is licensed under Creative Commons BY 3.0 <http://creativecommons.org/licenses/by/3.0>.

Activity 4: Canvas (Join the Dots)

 Project name (AppName)	COL_JoinTheDots
 App Name (Title)	Join The Dots
 App description (AboutScreen)	This app allows you to draw a picture by joining the dots on a Canvas component.
 Resources	https://goo.gl/JQMOLo
 Screenshot	

 <p>Expected Learning Outcomes</p>	<p>By the end of this activity you should be able to:</p> <ul style="list-style-type: none"> • Use the Canvas component • Set a Canvas BackgroundImage • Detect dragging of a finger on the screen
 <p>Components & Attributes</p>	<p> Canvas1</p> <ul style="list-style-type: none"> • BackgroundImage: pattern.png • Width: Fill parent • Height: Fill parent • PaintColor: Red <p> Button1</p> <ul style="list-style-type: none"> • Text: Clear

Blocks



```

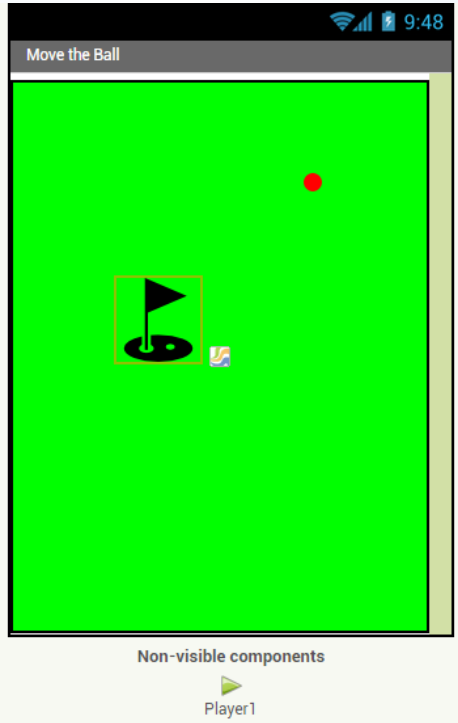
when Canvas1 .Dragged
  startX startY prevX prevY currentX currentY draggedAnySprite
do
  call Canvas1 .DrawLine
    x1 get prevX
    y1 get prevY
    x2 get currentX
    y2 get currentY

when Button1 .Click
do
  call Canvas1 .Clear
  
```

Walkthrough:

1. The **Canvas** component found under the **Drawing and Animation drawer** of the **Palette** is an important component when it comes to mobile apps. This is the component that facilitates creative aspects of an app, including game design. With the **gamification** of education taking centre stage in many forums, mastering the Canvas component will allow you to build apps that reflect the latest trends.
2. When considering the properties of the Canvas, make it fill the device screen so that the game play is in full screen. To achieve this, you will need to set the **Height** and **Width** properties to **Fill parent**. The other settings are **Automatic**, where the device determines the dimensions, or the **Custom setting**, should you specify the number of pixels. The Custom setting should be used sparingly as you want your app to be used on multiple devices of varying dimensions. The Fill parent setting ensures that your component fits any screen regardless of dimensions.
3. Loading a **BackgroundImage** for a **Canvas** is the same as loading an **Image** for a **Button**. The **PaintColor** is the colour of the lines you will be drawing on the Canvas. You can get more info by clicking on the “?”
4. The Dictation app taught you how to get results from a block that had **return values**. This one is no different. Just make sure you **don't click**.
5. Complete the block and test your app.

Activity 5: Ball and ImageSprite (Move the Ball)

 Project name (AppName)	COL_MoveTheBall
 App Name (Title)	Move The Ball
 App description (AboutScreen)	<p>This app uses a ball and an ImageSprite to simulate a golf game. The ball needs to be flung towards the hole. If the ball hits the hole the app will say “Hit.”</p>
 Resources	https://goo.gl/cfnhh1
 Screenshot	



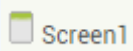
Expected Learning Outcomes

By the end of this activity you should be able to:

- Use the Canvas component
- Set a Canvas BackgroundImage
- Detect dragging of a finger on the screen

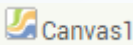


Components & Attributes



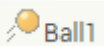
Screen1

- screenOrientation: **Portrait**



Canvas1

- BackgroundColor: **Green**
- Width: **Fill parent**
- Height: **Fill parent**



Ball1

- PaintColor: **Red**
- Radius: **7**



ImageSprite1

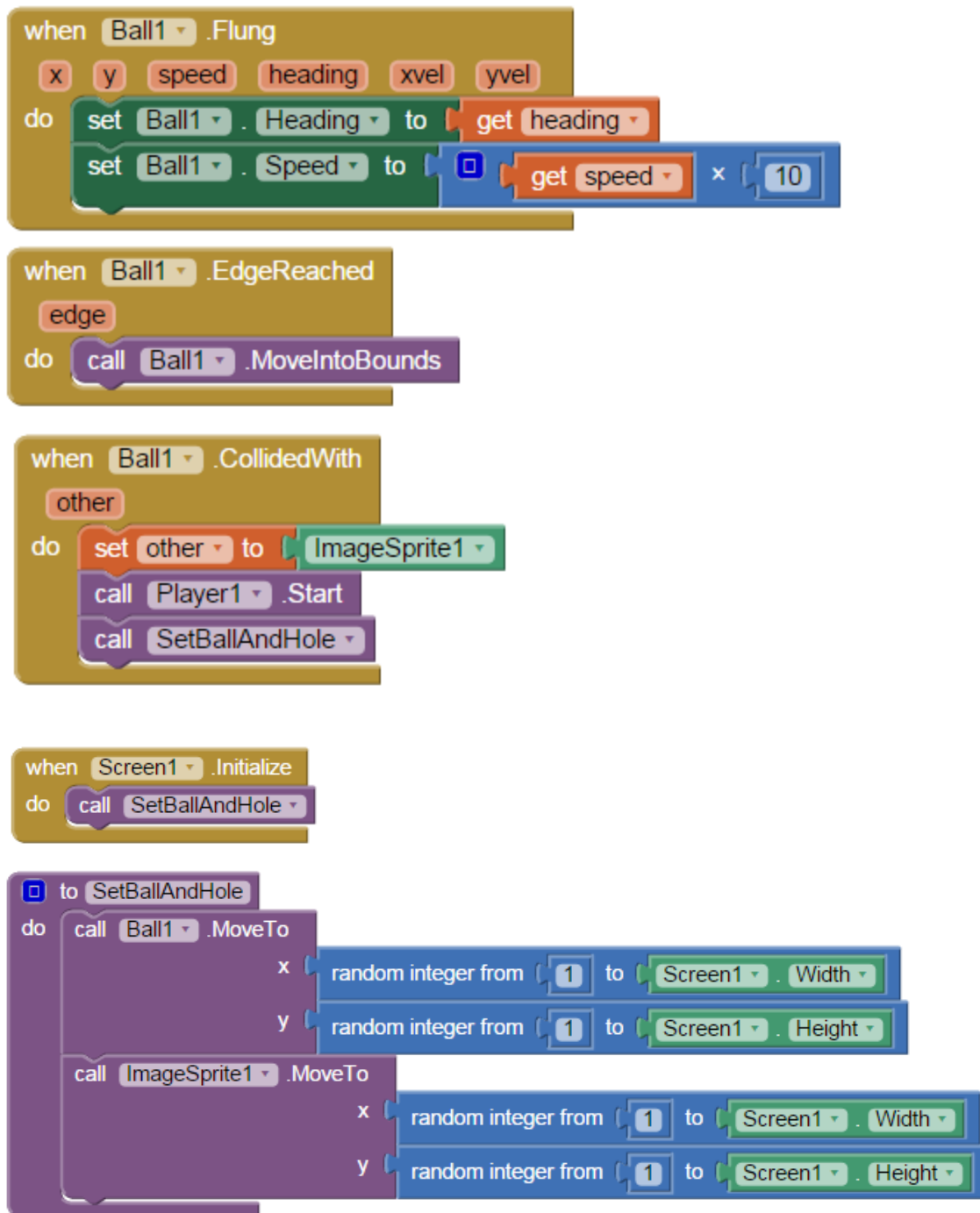
- Picture: **golf25.png**



Player1

- Source: **Sv-hit.ogg**

Blocks



Walkthrough:

1. The **ScreenOrientation** property of the **Screen** component determines the orientation of the app on the device screen. Usually it is set to **Unspecified**, which lets the device decide how the app is displayed on the screen according to the settings of the screen. The property **Sensor** specifically instructs the app to reorient itself based on the sensor direction of the device. The **User** property allows the user to select the orientation of the app. If you want to lock the orientation of an app, you can use the **Portrait** and **Landscape** properties to instruct the app to stay in one orientation regardless of the device orientation. In this case, you will be choosing **Portrait**.
2. The **Ball** component and the **ImageSprite** component are the key components used to build any animation or game on the AI2 platform. They are located in the **Drawing and Animation drawer** of the **Palette**. Change the **PaintColor** and **Radius** of the ball to suit the app. Set the **Picture** of **ImageSprite1** to **golf25.png** (in the Resources pack).
3. The **Player** component located in the **Media drawer** of the **Palette** is used to play back sound clips such as music or alert sounds. For shorter sound clips you can also use the **Sound** component. However, the Sound component is not compatible with all Android devices. Therefore, it is safer to always use the **Player** component in your apps. Set the **Source** property of **Player1** to **Sv-hit.ogg** (in the Resources pack).
4. This app has multiple aspects and cannot be explained in a single sentence. The description of the app will be as follows:

The ball and hole will appear in a random location on the screen when the app starts for the first time. The ball needs to be flung towards the hole. If the ball hits the hole the app will say "Hit." The location of the hole and the ball are re-set to random locations each time the ball hits the hole.

5. To build the logic blocks for the app you will need to break the app description into individual sentences as follows:

The ball and hole will appear in a random location on the screen when the app starts for the first time.

Since you will be resetting the location of the ball and the hole to random locations multiple times, it makes sense to create one logic block that can be reused. This is a core concept in object oriented programming (OOP). To achieve this, you will use a **[to Procedure do]** located in the **Procedures** section within the **Built-in blocks** (Figure 2.28). When you write large apps you will be using many Procedures. In order to keep track of them you need to name them in a sensible manner. Usually the name will reflect what the procedure will do. In this case, name the Procedure **SetBallAndHole** as shown in Figure 2.29.

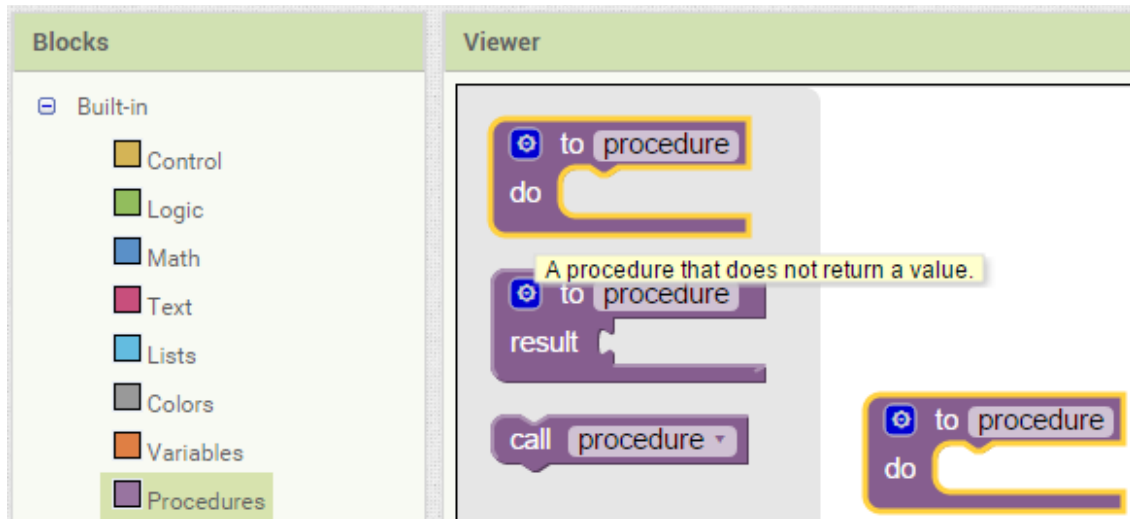


Figure 2.28 Creating a new procedure



Figure 2.29 Renaming a procedure

6. You now need to build the logic blocks for the **Procedure SetBallAndHole** as shown in Figure 2.30. Use the **[call Ball1.MoveTo]** block and **[call ImageSprite1.MoveTo]** blocks to set the locations of the **Ball** and the **ImageSprite**.

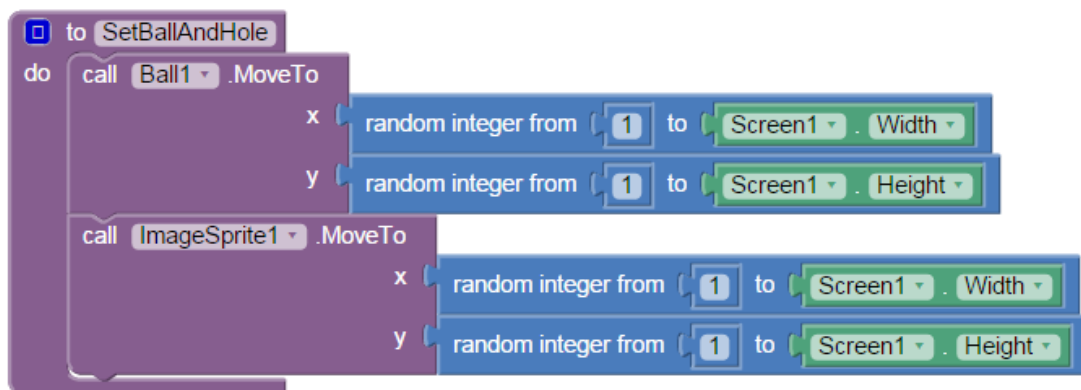


Figure 2.30 Procedure for randomly setting the positions of the Ball and the Hole

To set the X and Y coordinates to random values, use the **[random integer from xx to xx]** block located in the **Math** section of the **Built-in blocks** (Figure 2.31). Note how the blocks from each section of the **Built-in blocks** are colour-coded — the **Math blocks** are light blue, **Text blocks** maroon and **Procedure blocks** purple. This colour coding will help you to locate blocks easily within your app.

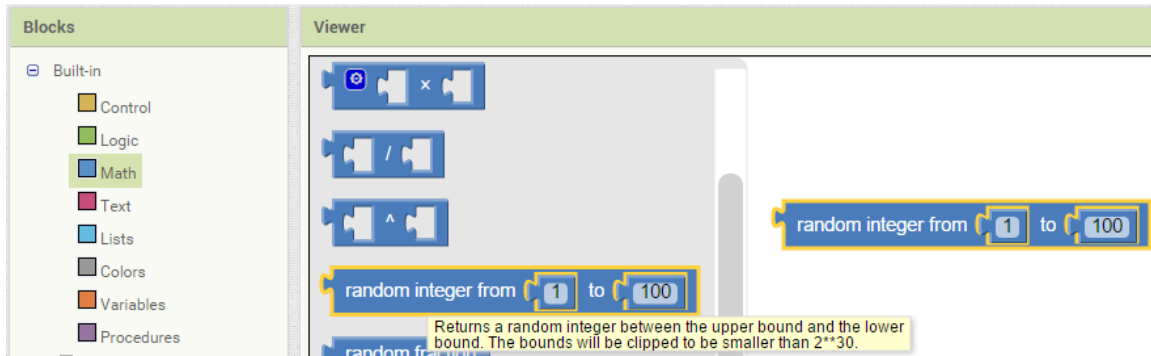


Figure 2.31 Using the random integer block in the Math section

The Ball and the Hole need to appear within the boundaries of the device screen. They will be invisible if they go beyond that. Since you are building the app for multiple devices with various screen sizes, you cannot define the bounds as an exact number. The coordinates (1,1) always represent the top left-hand corner of the device. You need to set the maximum X value and maximum Y value from there. To do this, click on **100** and press **DEL**. Fill the space with **Screen1 . Width** and **Screen1 . Height** blocks as shown in Figure 2.30. Now you have built your Procedure.

7. You need to set the locations of the Ball and Hole at app starting. This is achieved using the **[when Screen1.Initialize do]** block (Figure 2.32). This block is the first block executed when the app is launched. The **[call SetBallAndHole]** block is used to execute the **SetBallAndHole Procedure**. This block is located inside the **Procedures** section.

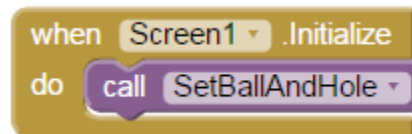


Figure 2.32 Calling a procedure at Screen Initialize

You have now built the logic for the sentence:

The ball and hole will appear in a random location on the screen when the app starts for the first time.

8. The second sentence is as follows:

The ball needs to be flung towards the hole.

You will use the **[when Ball1.Flung do]** block (Figure 2.33) to determine what happens to the ball when it is flung on the screen by the user's finger. Use the **return value** **get heading** to set the direction of the Ball.



Figure 2.33 The block to determine what happens when the Ball is flung by the user

The **Speed** at which the user flings the Ball can be set from the `get speed` return value. As this is a small number, use a **multiplication block** from the **Math section** to multiply the speed by a factor of 10 before setting it as the **Speed of the Ball**. This will ensure that the ball moves fluidly when the user flings it.

To stop the Ball at the edge of the screen, build the block shown in Figure 2.34. If you want the Ball to bounce back off the edge of the screen, build the logic block as shown in Figure 2.35.

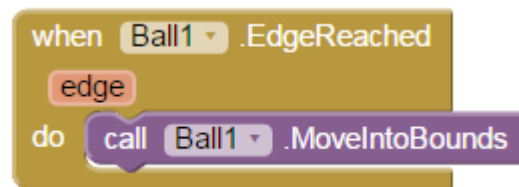


Figure 2.34 Moving the Ball back inside the bounds of the screen

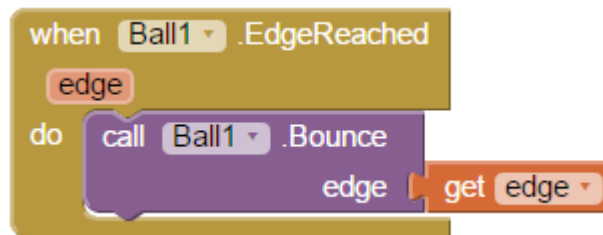


Figure 2.35 Making the Ball bounce back off of the edge of the screen

- Now you need to build the logic block for the third and final sentences:

If the ball hits the hole the app will say “Hit.” The location of the hole and the ball are re-set to random locations each time the ball hits the hole.

This is achieved using the **[when Ball1.CollidedWith do]** block. We need to check whether the Ball has collided with the Hole. This is determined by the **other** parameter of the block. Set it to **ImageSprite1**. The **Player** will play the sound clip once a collision takes place. Once the player has stopped, the **SetBallAndHole** procedure resets the locations.

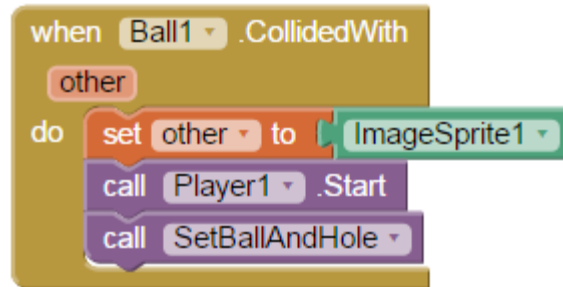


Figure 2.36 Detecting a collision between the Ball and the ImageSprite






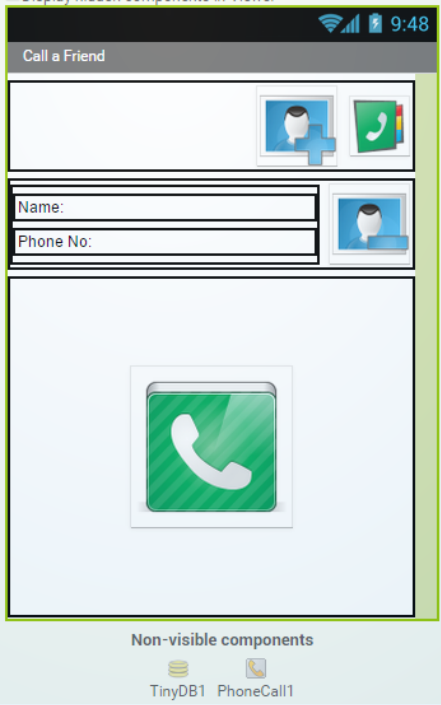
10. Congratulations! You have now mastered the **Canvas** component. Now it's time to play some golf.

In the Resources:

“golf25.png” by Yannick <http://yanlu.de> from www.flaticon.com is licensed under Creative Commons BY 3.0. <http://creativecommons.org/licenses/by/3.0>.

“Sv-hit.ogg” by M. Kihlstedt, N. Vion (The Shtooka Project) [CC BY 2.0 fr (<http://creativecommons.org/licenses/by/2.0/fr/deed.en>)], via Wikimedia Commons. <http://upload.wikimedia.org/wikipedia/commons/3/32/Sv-hit.ogg>.

Activity 6: TinyDB, PhoneNumberPicker and PhoneCall (Call a Friend)

 Project name (AppName)	COL_CallAFriend
 App Name (Title)	Call A Friend
 App description (AboutScreen)	This app uses the PhoneNumberPicker and a TinyDB to store a contact in a list and make a phone call to a contact from that list.
 Resources	https://goo.gl/LvFem0
 Screenshot	



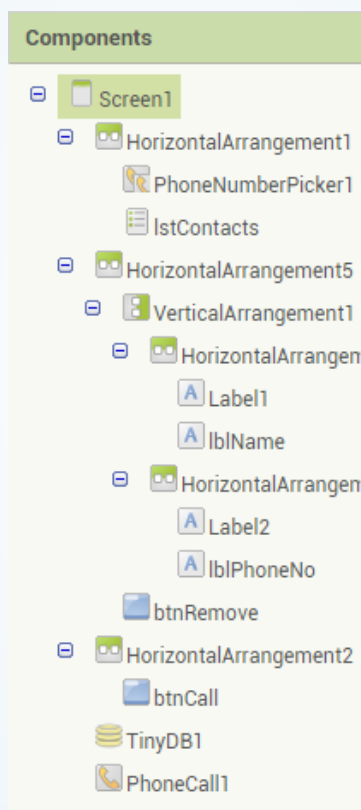
Expected Learning Outcomes


By the end of this activity you should be able to:

- Use nested Layout components to achieve a complex look and feel in the interface
- Use the PhoneNumberPicker to select a contact from the phone's address book
- Save, retrieve and delete tag-value pairs from the TinyDB
- Populate a ListPicker using a TinyDB
- Make a call using the PhoneCall component




Components & Attributes



 HorizontalArrangement1

- AlignHorizontal: **Right**
- Width: **Fill Parent**

 PhoneNumberPicker1

- Image: **picture-add-icon.png**



Components & Attributes (cont'd)

IstContacts

- Image: Address-Book-icon.png

Label1

- Text: Name

lblName

- Text:

Label2

- Text: Name

lblPhoneNo

- Text:

btnRemove

- Image: picture-remove-icon.png

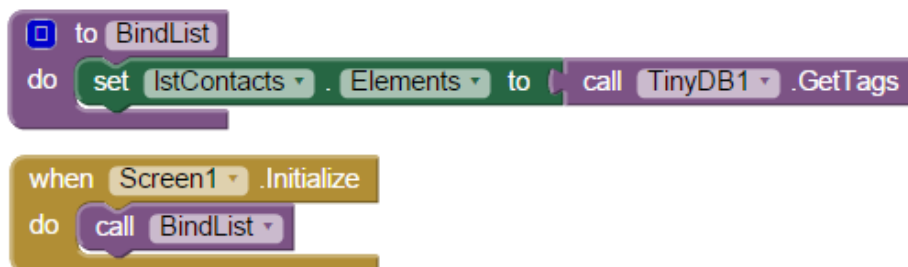
btnCall

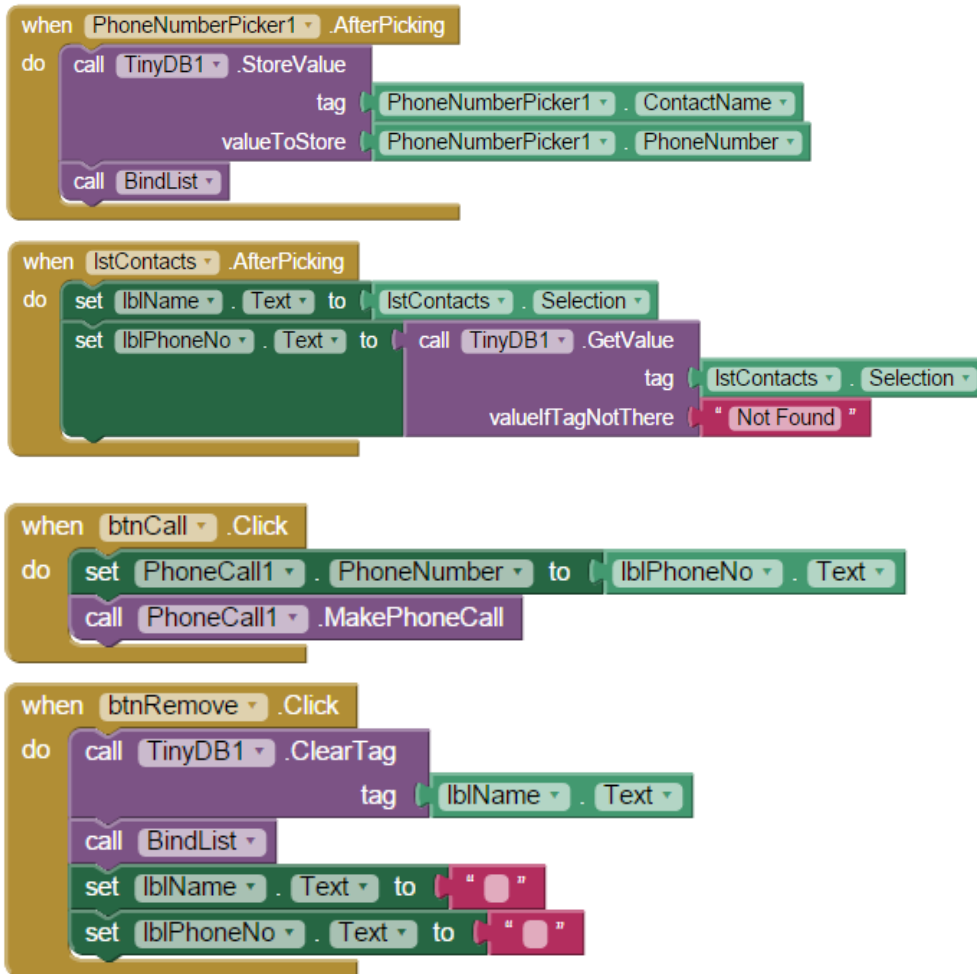
- Image: app-phone-icon.png

TinyDB1

PhoneCall1

Blocks





Walkthrough:

1. How your app looks on a user's phone determines most of the app's success, so it is very important to pay attention to its theme, icons, wording, colour scheme and layout. Your app also needs to be consistent — it should look the same on any phone or tablet regardless of the make, model or screen size. To achieve this consistency, you will be using the components found in the **Layout drawer** of the **Palette**. The **VerticalArrangement** component aligns other components vertically on the screen as shown in Figure 2.37. You can align the components within the **VerticalArrangement** by manipulating the **Properties** of the component as shown in Figure 2.38.

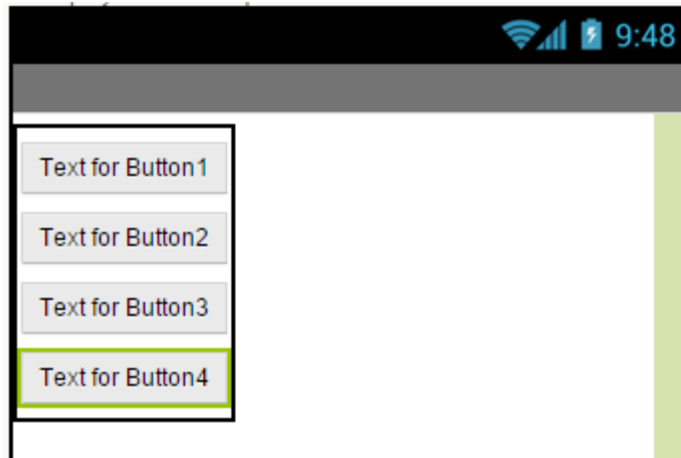


Figure 2.37 The VerticalArrangement component

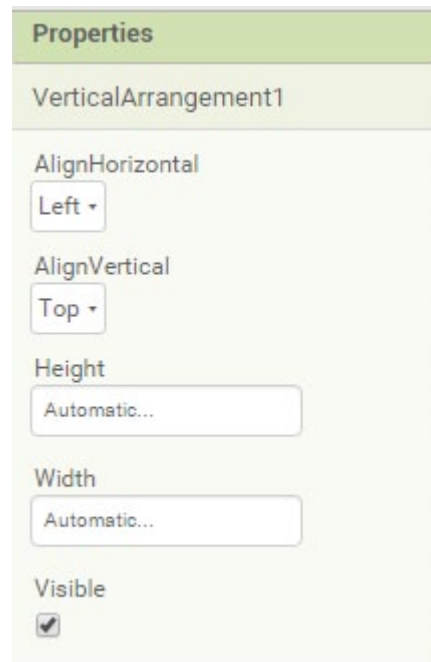



Figure 2.38 Properties of the VerticalArrangement component

The  **HorizontalArrangement** arranges the components horizontally across the screen as shown in Figure 2.39. As with the **VerticalArrangement**, you can align the components within the **HorizontalArrangement** by manipulating the **Properties** of the component.

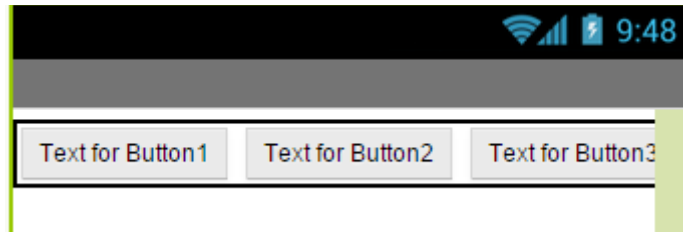


Figure 2.39 The *HorizontalArrangement* component

2. The **PhoneNumberPicker** and **PhoneCall** components are located inside the **Social drawer** of the **Palette**. The **PhoneNumberPicker** is used to pick the details of a contact from the phone's contacts list. The following properties can be manipulated once the contact has been picked from the list:

- **ContactName**: the contact's name
- **PhoneNumber**: the contact's phone number
- **EmailAddress**: the contact's email address
- **Picture**: the name of the file containing the contact's image, which can be used as a **Picture** property value for the **Image** or **ImageSprite** component.

More details about the **PhoneNumberPicker** can be found at <http://ai2.appinventor.mit.edu/reference/components/social.html#PhoneNumberPicker>

The **PhoneCall** component is a **non-visible component** that is used to dial out a particular number. It is normally used with a **PhoneNumberPicker** component. More details on the **PhoneCall** component can be found at <http://ai2.appinventor.mit.edu/reference/components/social.html#PhoneCall>

3. Another component you are encountering for the first time is the **Label** component inside the **User Interface drawer** of the **Palette**. Labels are widely used in apps to display static text. The **Text** property of the **Label** cannot be changed by the user (unless the app wants the user to do so). It is always set or changed by the app. Labels are normally used to describe a field — for example, what the user needs to enter into a **TextBox**.
4. Note that two of the **Labels** and the two **Buttons** have been renamed **lblName**, **lblPhoneNo**, **btnRemove** and **btnCall**. This makes it much easier for you to identify the components when you are building the logic blocks. To rename a component, select it from the **Components list** in the **Designer view**. Then press **Rename** at the bottom of the pane (Figure 2.40). Type the new name in the popup window and press **OK**. This action will rename the component and all associated blocks with the new name you have provided. Additionally, you can select a component and press **Delete** at the bottom of the pane to remove it from your project.



IMPORTANT: Be careful when you do this as it will also delete all associated logic blocks you have built.

NOTE: When renaming, prefixes such as **lbl** — short for “Label” — and **btn** — short for “Button”— are used. These are standard naming conventions used in programming. It is good practice to follow these conventions in your apps.

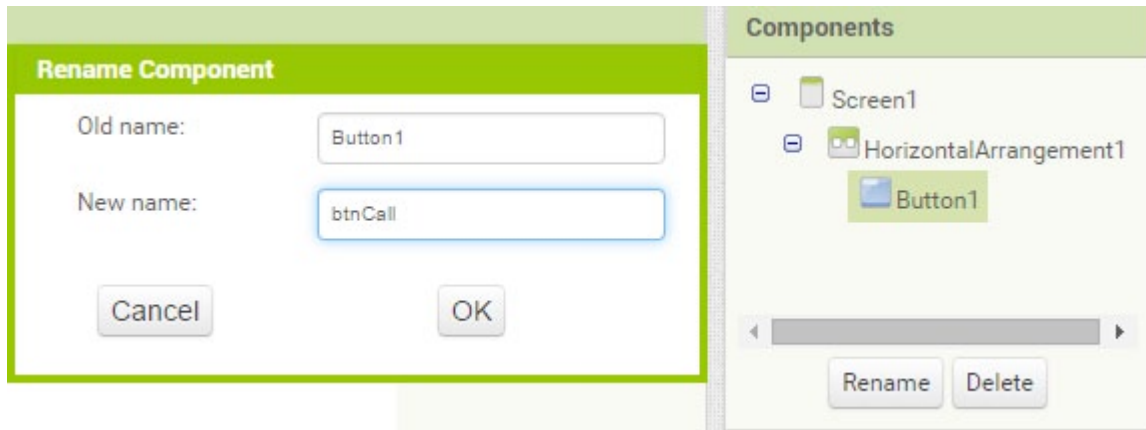


Figure 2.40 Renaming a component

- When you are developing your apps, you will find a constant need to store data in a database that can be accessed later. If you store a value using a **Variable** block built using the blocks inside the **Variables** section of the **Built-in blocks** in **Blocks** view, you will only be able to use that value during the execution of your app. Say, for example, you want to store the highest score of a game you have developed and display it. You can do this using a **Variable** block such as the one shown in Figure 2.41. This will work fine until you are running the app. However, once you exit the app the high score value is lost — that is, the next time you open the app, the high score value would be initialised to 0.

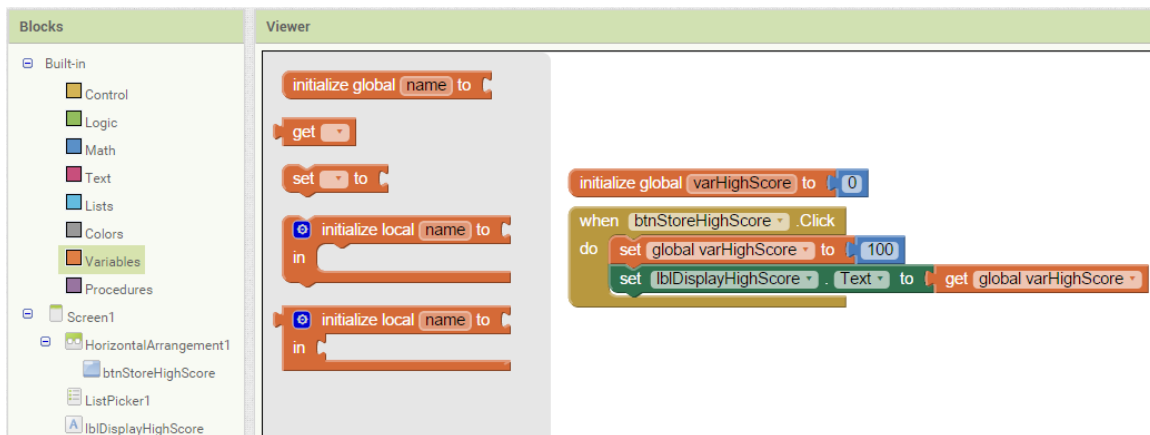


Figure 2.41 Using variables to store and display a value

To be able to recall a value even after the app has been restarted, you will need to use a database. There are four different types of non-visible components available for storing information under the **Storage drawer** of the **Palette**. The components are as follows:

- **File**: Non-visible component for storing and retrieving files. Use this component to write or read files on your device. The default behaviour is to write files to the private data directory associated with your app. The **Companion** writes files to **/sdcard/AppInventor/data** for easy debugging. If the file path starts with a slash (/), then the file is created relative to **/sdcard**. For example, writing a file to **/myFile.txt** will write the file in **/sdcard/myFile.txt**.
 - For more information on this, see <http://ai2.appinventor.mit.edu/reference/components/storage.html#File>
- **FusionTablesControl**: A non-visible component that communicates with Google Fusion Tables. Fusion Tables let you store, share, query and visualise data tables; this component lets you query, create and modify these tables.
 - For more information on this, see <http://ai2.appinventor.mit.edu/reference/components/storage.html#FusionTablesControl>
- **TinyDB**: Non-visible component that stores data for an app. TinyDB is a persistent data store for the app. The data stored in a TinyDB will be available every time the app is run. Data items are strings stored under tags. To store a data item, you specify the tag it should be stored under. You can then retrieve the data that was stored under a given tag.
 - For more information on this, see <http://ai2.appinventor.mit.edu/reference/components/storage.html#TinyDB>
- **TinyWebDB**: Non-visible component that communicates with a Web service to store and retrieve information.
 - For more information on this, see <http://ai2.appinventor.mit.edu/reference/components/storage.html#TinyWebDB>

This toolkit only uses the **TinyDB** component, which is the most commonly used component for storing data in the persistent storage component of the mobile device. You will use the other storage components when you start developing complex applications that require real-time data.

6. The **TinyDB** component stores information using a **Tag-Value pair** concept. In this particular app, we are using the TinyDB to store the **Name** and **Phone Number** of a contact — that is, the **Tag** will be the **Name** and the **Value** will be the **Phone Number**. Say, for example, we want to store John's phone number. The Tag-Value pair would be **John-5552223331**.

The **[call TinyDB1.StoreValue]** block (procedure) is used to store a Tag-Value pair in the TinyDB as shown in Figure 2.32.

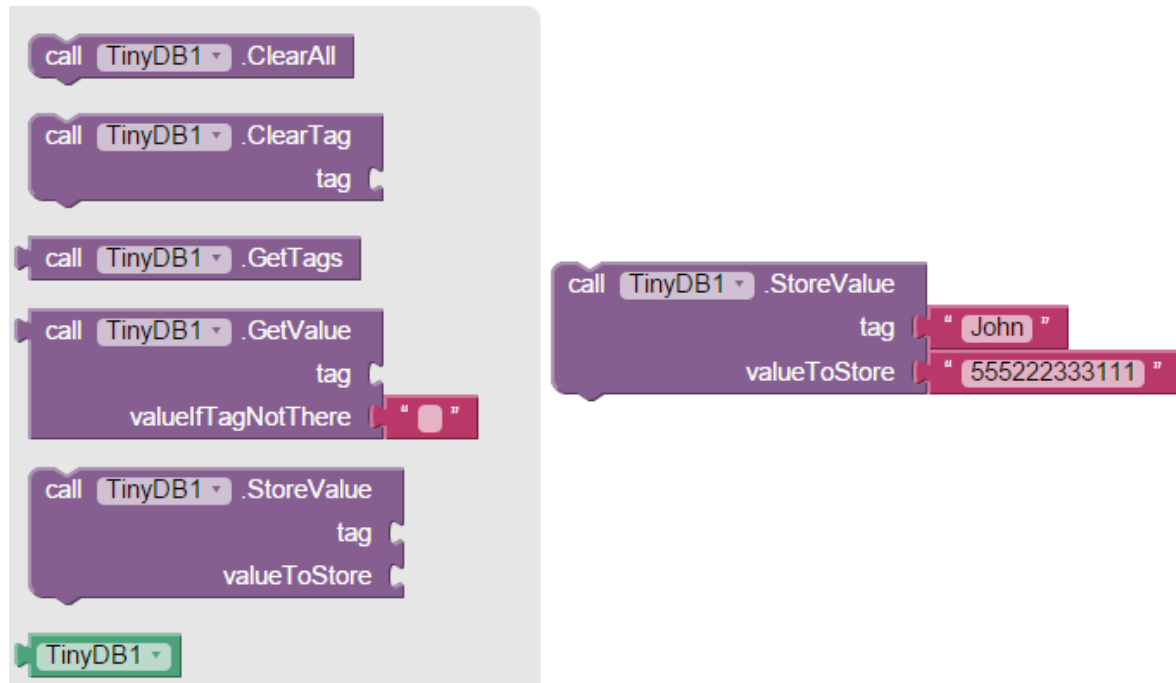


Figure 2.42 Inserting a Tag-Value pair into a TinyDB

In this app, the user will pick a contact from the list of contacts on the phone using the **PhoneNumberPicker** component. The information of the contact picked is captured using the **[when PhoneNumberPicker1.AfterPicking do]** block, which is executed after the user picks a contact from the list. At this point, the name and the phone number are stored inside the **ContactName** and **PhoneNumber** properties of the **PhoneNumberPicker**. These will be added to the TinyDB for storage as shown in Figure 2.43. It also shows the **Procedure BindList**. This is discussed in the next section.

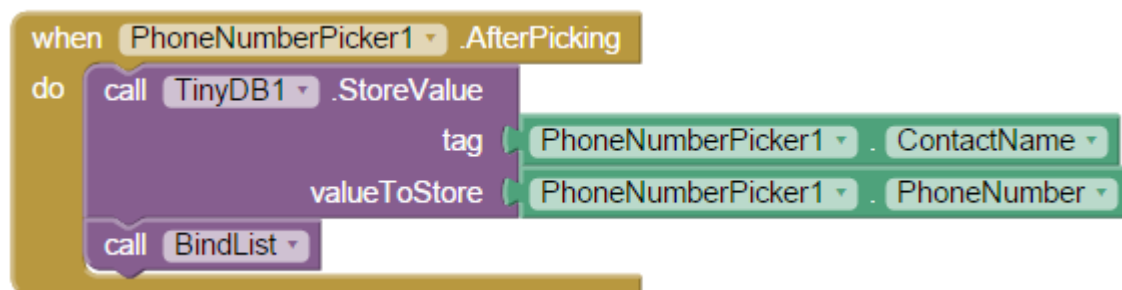


Figure 2.43 Inserting a contact from the PhoneNumberPicker into a TinyDB

To remove an entry from the TinyDB, use the **[call TinyDB.ClearTag]** procedure as shown in Figure 2.44.



Figure 2.44 Removing a Tag-Value pair from the TinyDB

7. Once you have picked a contact from the contacts list on your phone and saved the contact in the TinyDB, you will need to view the contacts you have saved in order to make a call. To do this, you will use the **ListPicker** component located in the **User Interface drawer** of the **Palette**. There is another component called **ListView** in the **User Interface drawer** which can hold a list. However, this list is displayed on the app screen permanently whereas the ListPicker only displays the list when it's clicked. You can use the two components depending on the requirements of your app. For the purposes of this app, you will be using the ListPicker. More information about the ListView can be found at <http://ai2.appinventor.mit.edu/reference/components/userinterface.html#ListView>

Typically, you will be searching for a contact by name — that is, the list will need to display the list of names that have been saved into the TinyDB. The **Name** is saved as the **Tag** of the **Tag-Value pair**. Therefore, you will need to load all the Tags into the ListPicker, which is renamed **IstContacts**. This is achieved using a custom procedure called BindList as shown in Figure 2.45.



Figure 2.45 Populating the ListPicker with the Tags in the TinyDB

8. Congratulations! You can now store a Tag-Value pair inside the persistent storage of the device. This will allow you to build significantly sophisticated apps.






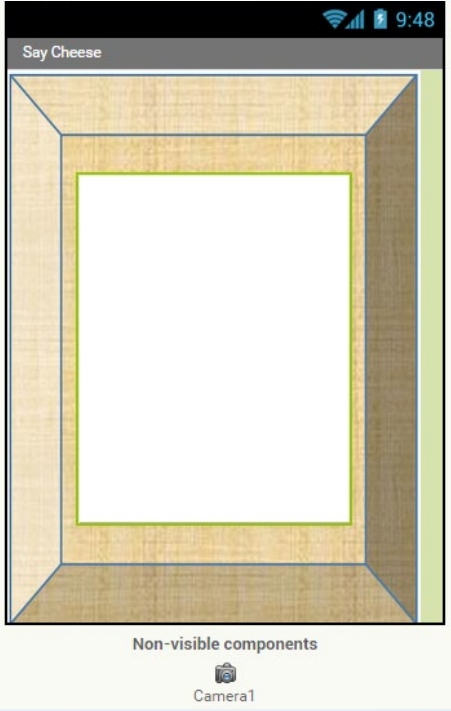
In the Resources:

“app-phone-icon.png” by Double-J Design is used under a CC BY 4.0 International License from www.iconarchive.com/show/apple-festival-icons-by-double-j-design/app-phone-icon.html.

“picture-add-icon.png” is used under a public domain licence from www.iconarchive.com/show/blue-bits-icons-by-icojam/picture-add-icon.html.

“Address-Book-icon.png” by Double-J Design is used under a CC BY 4.0 International Licence from “picture-remove-icon.png” is used under a public domain licence from www.iconarchive.com/show/blue-bits-icons-by-icojam/picture-remove-icon.html.

Activity 7: Camera (Say Cheese!)

 Project name (AppName)	COL_SayCheese
 App Name (Title)	Say Cheese!
 App description (AboutScreen)	<p>This app uses the camera component to take a picture by clicking on the empty area of the picture frame. The picture is then displayed in a frame.</p>
 Resources	https://goo.gl/cMJrha
 Screenshot	



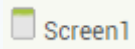
Expected Learning Outcomes

By the end of this activity you should be able to:

- Use the Camera component to take a picture
- Manipulate the Screen orientation of the app
- Manipulate the Background colour of the Screen
- Apply a Background Image to the Screen
- Change the Background colour of a Button
- Set the Background Image of a Button using blocks



Components & Attributes



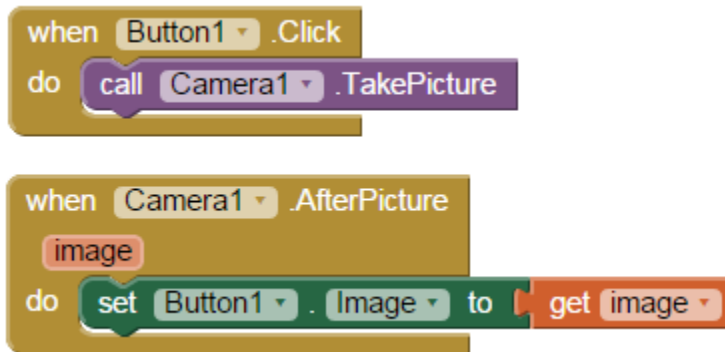
- AlignHorizontal: **Center**
- AlignVertical: **Center**
- BackgroundColor: **None**
- BackgroundImage: **imgPictureFrame.png**
- ScreenOrientation: **Portrait**



- Text:
- BackgroundColor: **White**
- Width: **210 pixels**
- Height: **270 pixels**








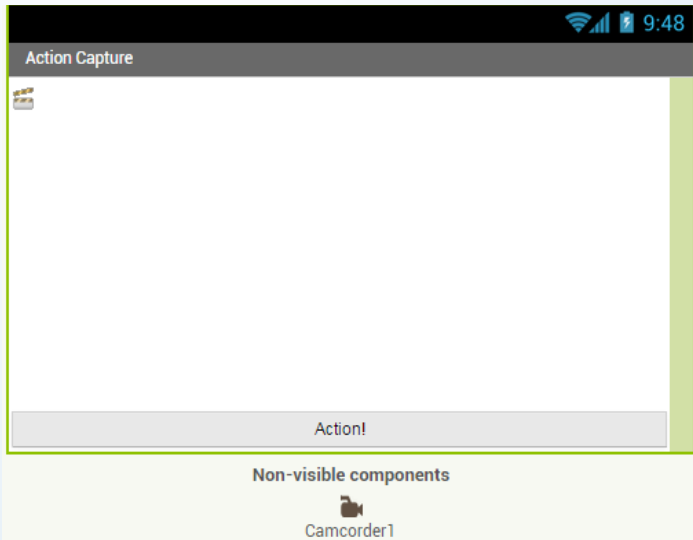
Blocks





Walkthrough:

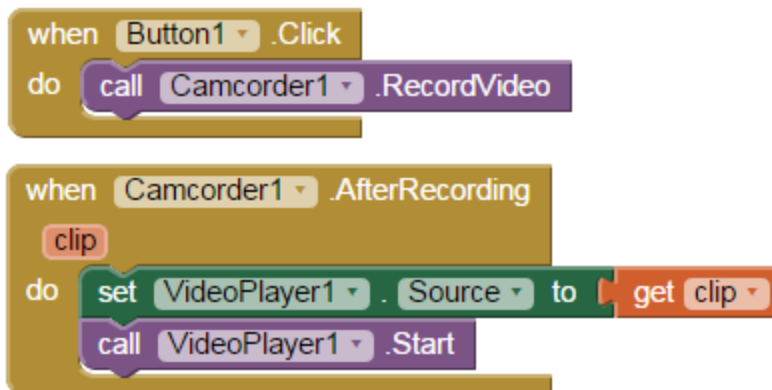
1. If you have reached this far, you will have a good grasp of app development on the AI2 platform. In this app you will be using the **Camera** component located in the **Media drawer** of the **Palette**.
2. This is pretty straightforward app considering how much you have already learned from the ones you have already mastered. You can make it into a selfie app by manipulating the properties of the Camera component.

Activity 8: Camcoder and VideoPlayer (Action Capture)

 Project name (AppName)	COL_ActionCapture
 App Name (Title)	Action Capture
 App description (AboutScreen)	This app uses the Camcorder component to capture a video clip. Once the clip has been captured, it is played back using the VideoPlayer component.
 Resources	https://goo.gl/liL46j
 Screenshot	

 Expected Learning Outcomes	<p>By the end of this activity you should be able to:</p> <ul style="list-style-type: none"> • Use the Camcorder component to capture video • Use the VideoPlayer component to play back a video clip
 Components & Attributes	<div data-bbox="646 457 776 499">Screen1</div> <ul style="list-style-type: none"> • AlignHorizontal: Center • AlignVertical: Center • ScreenOrientation: Landscape <div data-bbox="646 714 776 756">Button1</div> <ul style="list-style-type: none"> • Text: Action! • Width: Fill parent <div data-bbox="646 924 812 966">Camcorder1</div> <div data-bbox="646 976 820 1018">VideoPlayer1</div>

Blocks



Walkthrough:






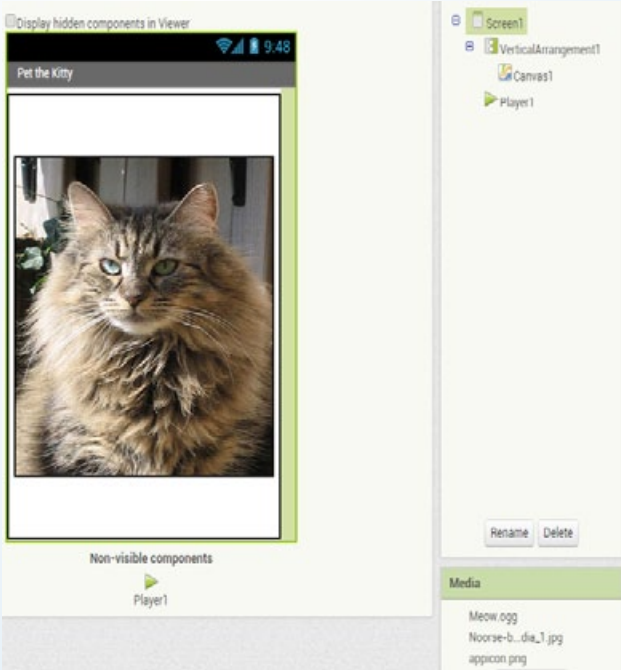
1. This app uses the **Camcoder** and **VideoPlayer** components of the **Media drawer** in the **Palette**. It is similar to the app you built in Activity 7.
2. You can now see that AI2 allows you to use many of the built-in features and software of the Android device. To use other software and hardware on your device from within your app, you will need to use the non-visible ⚡ **ActivityStarter** component from the **Connectivity drawer** of the **Palette**. More details on how to use the **Activity Starter** can be found at <http://ai2.appinventor.mit.edu/reference/components/connectivity.html#ActivityStarter>

You will be able to find lots more information, examples, code snippets, etc., at Pura Vida Apps <http://puravidaapps.com>. You can find answers to your questions at the MIT App Inventor Support Forum at <https://groups.google.com/forum/#!forum/mitappinventortest>.

You will now be able to do the Tutorials on your own. Make sure you do them sequentially and do them all. By the end of the Tutorials, you will be able to design and develop really cool Android apps for your teaching, learning and even commercial purposes.

3. Tutorials

Tutorial 1: Pet the Kitty

 Project name (AppName)	COL_PetTheKitty
 App Name (Title)	Pet The Kitty
 App description (AboutScreen)	<p>This is a virtual pet application. An image of a cat appears in the centre of the screen and makes a purring sound when stroked with a finger. In addition, the phone vibrates to the purring sound.</p>
 Resources	https://goo.gl/E6GvQg
 Screenshot	 <p>The screenshot displays the 'Pet the Kitty' app interface. At the top, there's a status bar with the time 9:48. Below it, the app title 'Pet the Kitty' is visible. The main content area features a large image of a fluffy brown cat. At the bottom, there's a 'Non-visible components' section with a 'Player1' icon. On the right side, there's a 'Media' section with a list of files: 'Meow.ogg', 'Noorse-b...dia_1.jpg', and 'appicon.png'. There are also 'Rename' and 'Delete' buttons above the media list.</p>




Expected Learning Outcomes

By the end of this activity you should be able to:


- Use arrangement components to organise the screen
- Design user experiences (UX) with images, sound, touch and vibration




Components & Attributes

 **Screen1**

- Icon: **appicon.png**
- ScreenOrientation: **Portrait**

 **VerticalArrangement1**

- Width: **Fill parent**
- Height: **Fill parent**
- AlignHorizontal: **Center**
- AlignVerticle: **Center**

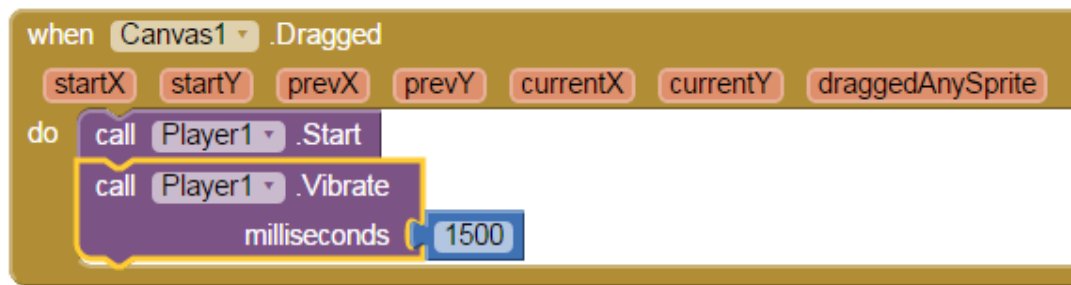
 **Canvas1**

- BackgroundImage: **Noorse-boskat_wikipedia_1.png**
- Width: **300 pixels**
- Height: **300 pixels**

 **Player1**

- Source: **Meow.ogg**

Blocks



In the Resources:

“Noorse-boskat wikipedia 1” by Wieke de Rijk, Netherlands at nl.wikipedia. Licensed under CC BY 2.5 via Wikimedia Commons: http://commons.wikimedia.org/wiki/File:Noorse-boskat_wikipedia_1.JPG#mediaviewer/File:Noorse-boskat_wikipedia_1.JPG.

“Meow.ogg” by Dcrosby at en.wikipedia [CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>)], from Wikimedia Commons: <http://upload.wikimedia.org/wikipedia/commons/6/62/Meow.ogg>.

Tutorial 2: Swat the Mosquito

 Project name (AppName)	COL_SwatTheMosquito
 App Name (Title)	Swat The Mosquito
 App description (AboutScreen)	<p>This is an interactive game. The objective is to swat the mosquito with your finger. The mosquito flies randomly on the screen and the phone vibrates as the mosquito flies. When you swat the mosquito the game will say “Hit” and your score will increase by +1. You can reset the score to start a new game.</p>
 Resources	https://goo.gl/EFy0Pg
 Screenshot	



Expected Learning Outcomes

By the end of this activity you should be able to:

- Use image sprites
- Create and call procedures
- Set up and use global variables
- Work with the Clock component



Components & Attributes

Screen1

- Icon: **appicon.png**
- ScreenOrientation: **Portrait**

Canvas1

- BackgroundImage: **Brown_brick_wall.png**
- Width: **Fill parent**
- Height: **Fill parent**

isMosquito

- Picture: **Mosquito.png**
- Width: **35 pixels**
- Height: **35 pixels**

HorizontalArrangement1

- Width: **Fill parent**
- AlignHorizontal: **Right**

lblScoreTitle

- FontBold: **True**
- FontSize: **18.0**
- Text: **Score : -**

lblScoreCounter

- FontBold: **True**
- FontSize: **18.0**
- Text: **0**

btnResetScore

- BackgrounColor: **Yellow**
- FontSize: **18.0**
- Shape: **rounded**
- Text: **RESET**

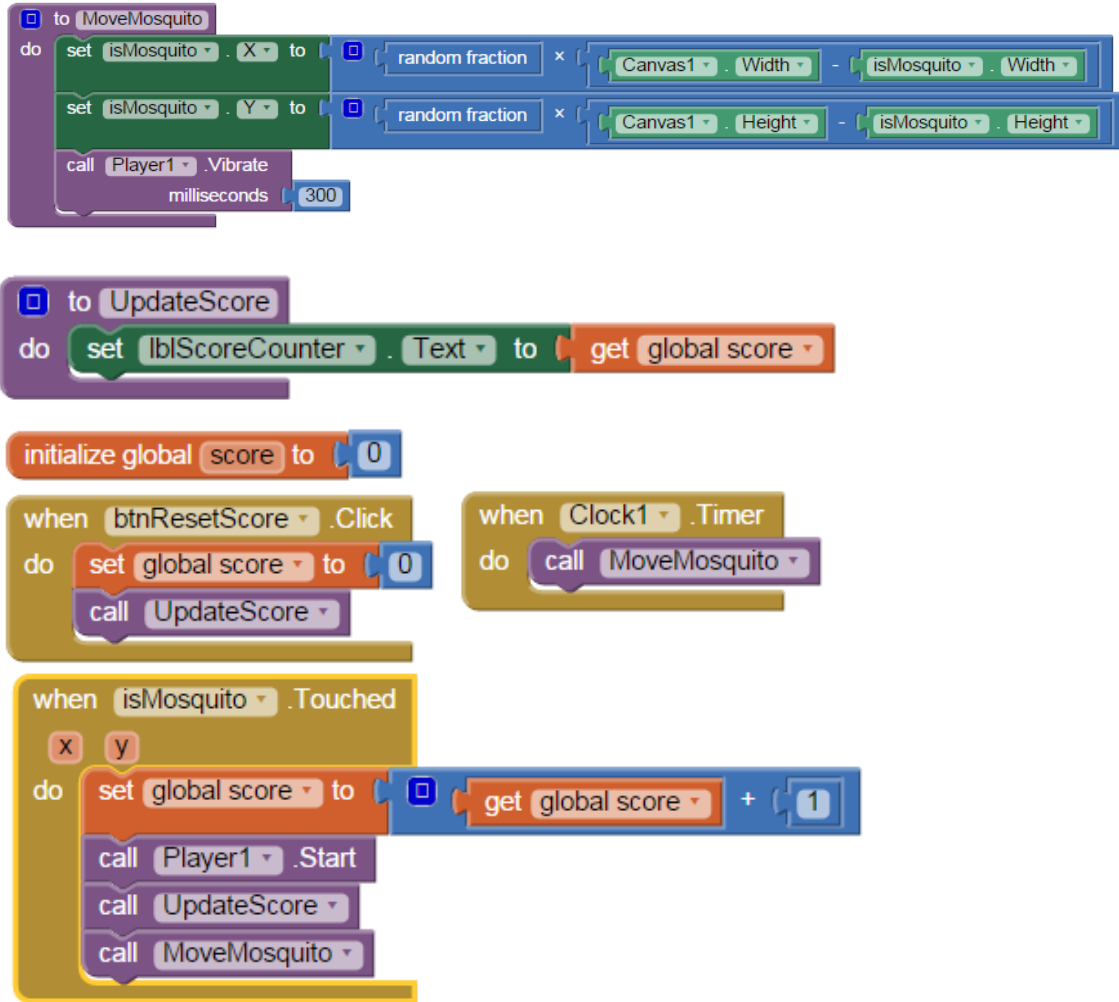
Clock1

- TimeInterval: **700**

Player1

- Source: **Sv-hit.ogg**

Blocks







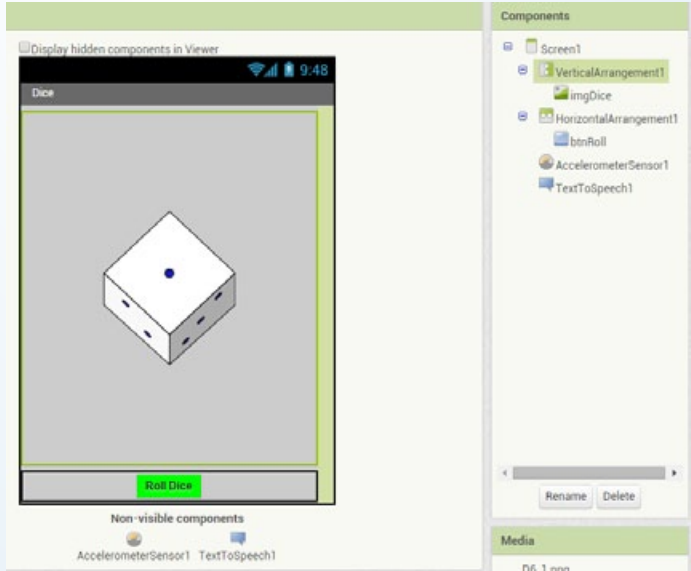
In the Resources:

“Mosquito.png” is based on the icon by www.freepik.com used under a CC BY 3.0 licence from www.flaticon.com.

“Brown brick wall” by Titus Tschardt: www.public-domain-image.com/free-images/textures-and-patterns/wall-texture/brick-wall-texture-pictures/brown-brick-wall-605x544.jpg Licensed under Public Domain via Wikimedia Commons: http://commons.wikimedia.org/wiki/File:Brown_brick_wall.jpg#mediaviewer/File:Brown_brick_wall.jpg.

“Sv-hit.ogg” by M. Kihlstedt, N. Vion (The Shtooka Project) [CC BY 2.0 from (<http://creativecommons.org/licenses/by/2.0/fr/deed.en>)], via Wikimedia Commons: <http://upload.wikimedia.org/wikipedia/commons/3/32/Sv-hit.ogg>.

Tutorial 3: Dice

 Project name (AppName)	COL_Dice
 App Name (Title)	Dice
 App description (AboutScreen)	<p>This app can be used as a replacement for a physical dice when playing board games such as Snakes and Ladders. Swing the phone to roll the dice. The app will speak out the number you have rolled.</p>
 Resources	https://goo.gl/s8GXMD
 Screenshot	




Expected Learning Outcomes

By the end of this tutorial you should be able to:


- Use TextToSpeech in a real-world scenario
- Use the AccelerometerSensor
- Use the control statement If-Then-Else to make selections
- Generate random numbers within a given range
- Assign media through blocks
- Pass parameters to procedures




Components & Attributes

 **Screen1**

- Icon: **appicon.png**
- ScreenOrientation: **Portrait**

 **VerticalArrangement1**


- Width: **Fill parent**
- Height: **Fill parent**
- AlignHorizontal: **Center**
- AlignVerticle: **Center**

 **imgDice**


- Picture: **D6_1.png**


 **HorizontalArrangement1**

- Width: **Fill parent**
- AlignHorizontal: **Center**

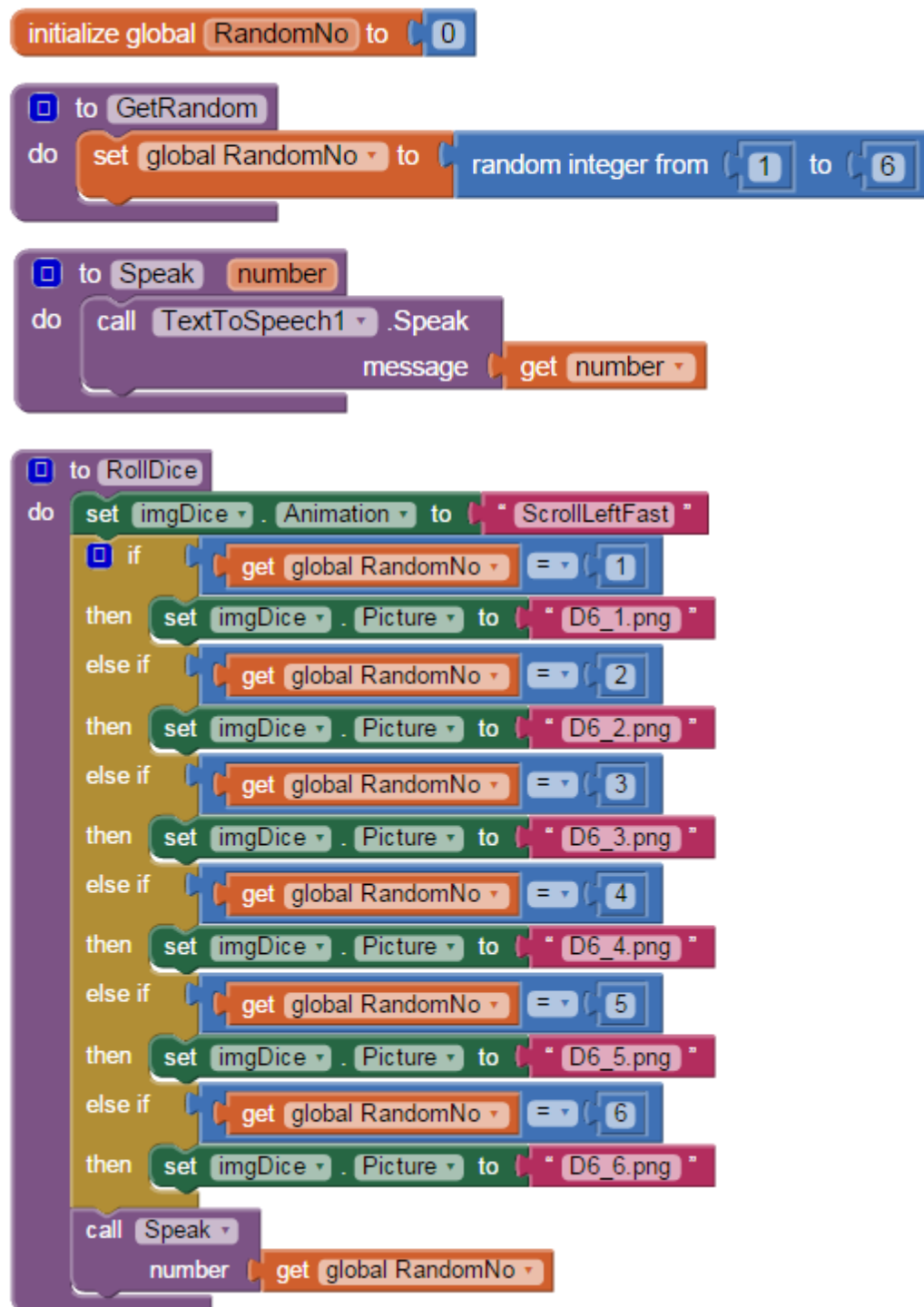
 **btnRoll**

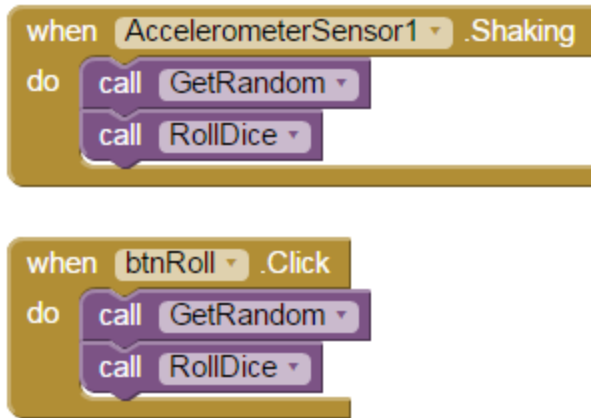
- BackgroundColor: **Green**
- Text: **Roll Dice**

 **AccelerometerSensor1**

 **TextToSpeech1**

Blocks





In the Resources:

“D6 1” by Christophe Dang Ngoc Chan (cdang). Own work. This vector image was created with Inkscape. Licensed under CC BY-SA 3.0 via Wikimedia Commons: http://commons.wikimedia.org/wiki/File:D6_1.svg#mediaviewer/File:D6_1.svg.

“D6 2” by Christophe Dang Ngoc Chan (cdang). Own work. This vector image was created with Inkscape. Licensed under CC BY-SA 3.0 via Wikimedia Commons: http://commons.wikimedia.org/wiki/File:D6_2.svg#mediaviewer/File:D6_2.svg.






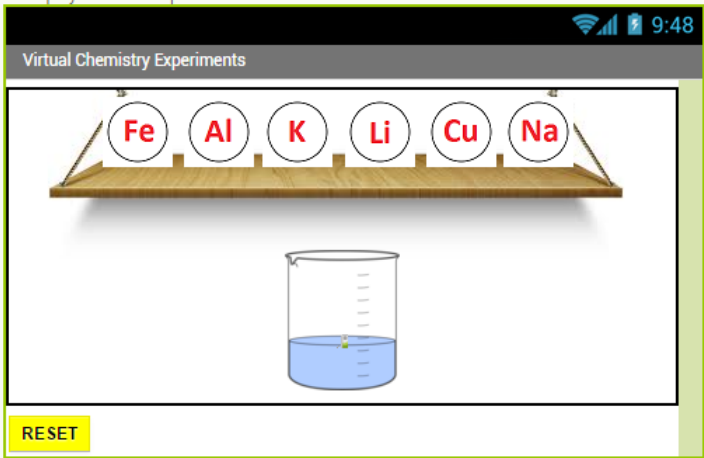

“D6 3” by Christophe Dang Ngoc Chan (cdang). Own work. This vector image was created with Inkscape. Licensed under CC BY-SA 3.0 via Wikimedia Commons: http://commons.wikimedia.org/wiki/File:D6_3.svg#mediaviewer/File:D6_3.svg.

“D6 4” by Christophe Dang Ngoc Chan (cdang). Own work. This vector image was created with Inkscape. Licensed under CC BY-SA 3.0 via Wikimedia Commons: http://commons.wikimedia.org/wiki/File:D6_4.svg#mediaviewer/File:D6_4.svg.

“D6 5” by Christophe Dang Ngoc Chan (cdang). Own work. This vector image was created with Inkscape. Licensed under CC BY-SA 3.0 via Wikimedia Commons: http://commons.wikimedia.org/wiki/File:D6_5.svg#mediaviewer/File:D6_5.svg.

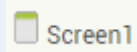
“D6 6” by Christophe Dang Ngoc Chan (cdang). Own work. This vector image was created with Inkscape. Licensed under CC BY-SA 3.0 via Wikimedia Commons: http://commons.wikimedia.org/wiki/File:D6_6.svg#mediaviewer/File:D6_6.svg.

Tutorial 4: Virtual Chemistry Experiments

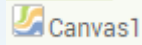
 Project name (AppName)	COL_VirtualChemistryExperiments
 App Name (Title)	Virtual Chemistry Experiments
 App description (AboutScreen)	This app uses the Canvas and ImageSprite components to demonstrate a virtual chemistry experiment. The user can drag and drop six different elements from the shelf into the beaker of water to see how they react.
 Resources	https://goo.gl/e91OBV
 Screenshot	
 Expected Learning Outcomes	<p>By the end of this tutorial you should be able to:</p> <ul style="list-style-type: none"> • Use global variables to hold components • Pass parameters into procedures • Reset X and Y coordinates of components using blocks



Components & Attributes



- Icon: **appicon.png**
- ScreenOrientation: **Landscape**



- Width: **Fill parent**
- Height: **Fill parent**



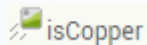
- Image: **imgShelf.jpg**



- Image: **imgPotassium.png**



- Image: **imgLithium.png**



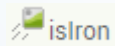
- Image: **imgCopper.png**



- Image: **imgAluminium.png**



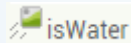
- Image: **imgSodium.png**



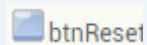
- Image: **imgIron.png**



- Image: **imgBeaker.png**

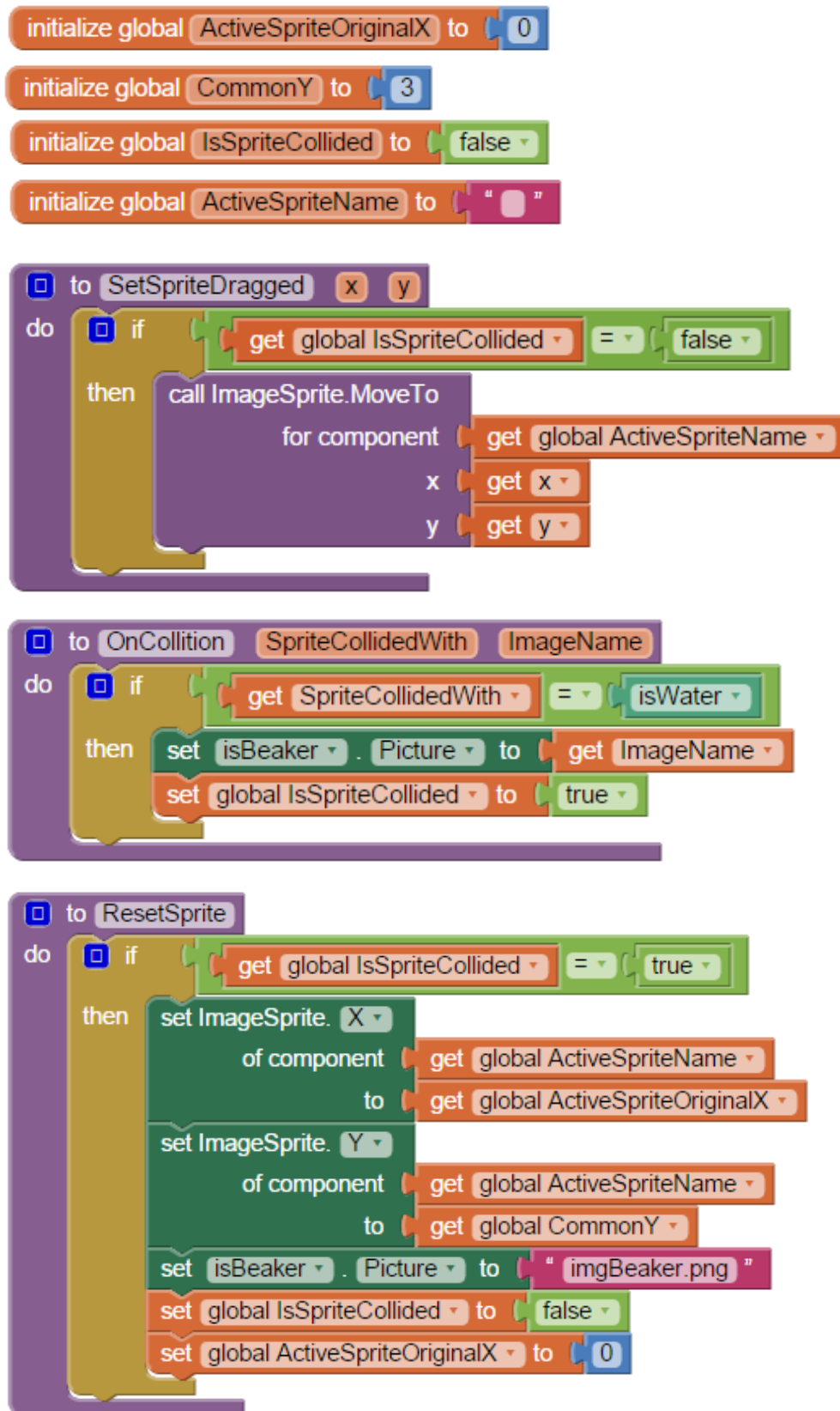


- Width: **10 pixels**
- Height: **10 pixels**



- BackgroundColor: **Yellow**
- FontBold: **True**
- Text: RESETBackgroundColor: **Yellow**
- FontBold: **True**
- Text: **RESET**

Blocks



```

when btnReset.Click
do call ResetSprite

```

```

when isIron.CollidedWith
other
do call OnCollition
SpriteCollidedWith get other
ImageName "imgBeaker.png"

```

```

when isIron.Dragged
startX startY prevX prevY currentX currentY
do set global ActiveSpriteName to isIron
set global ActiveSpriteOriginalX to 65
call SetSpriteDragged
x get currentX
y get currentY

```

```

when isAluminium.CollidedWith
other
do call OnCollition
SpriteCollidedWith get other
ImageName "imgBeaker.png"

```

```

when isAluminium.Dragged
startX startY prevX prevY currentX currentY
do set global ActiveSpriteName to isAluminium
set global ActiveSpriteOriginalX to 123
call SetSpriteDragged
x get currentX
y get currentY

```



```

when isPotasium .CollidedWith
  other
do
  call OnCollition
  SpriteCollidedWith get other
  ImageName "imgKReaction.jpg"

```

```

when isPotasium .Dragged
  startX startY prevX prevY currentX currentY
do
  set global ActiveSpriteName to isPotasium
  set global ActiveSpriteOriginalX to 179
  call SetSpriteDragged
    x get currentX
    y get currentY

```

```

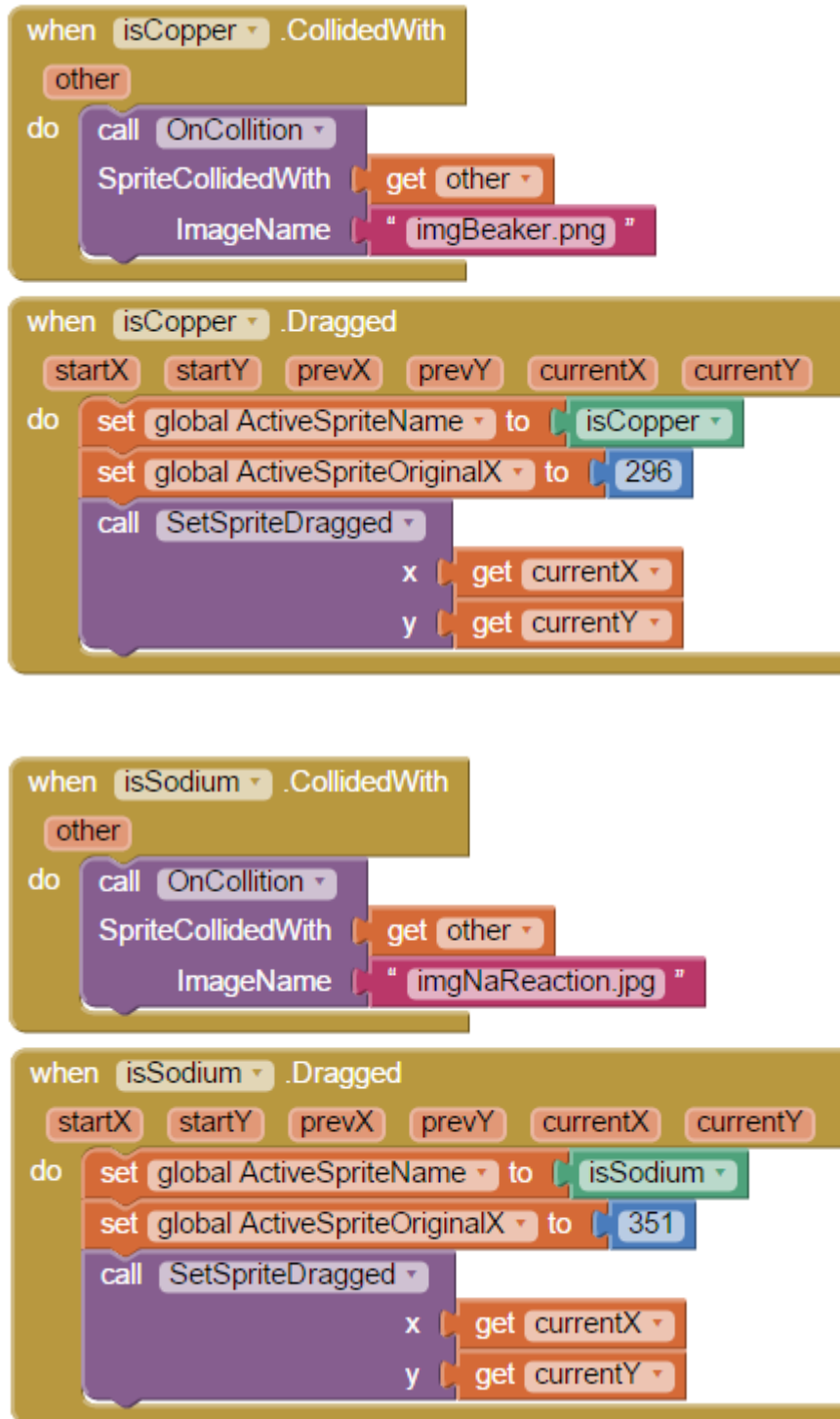
when isLithium .CollidedWith
  other
do
  call OnCollition
  SpriteCollidedWith get other
  ImageName "imgLiReaction.jpg"

```

```

when isLithium .Dragged
  startX startY prevX prevY currentX currentY
do
  set global ActiveSpriteName to isLithium
  set global ActiveSpriteOriginalX to 238
  call SetSpriteDragged
    x get currentX
    y get currentY

```






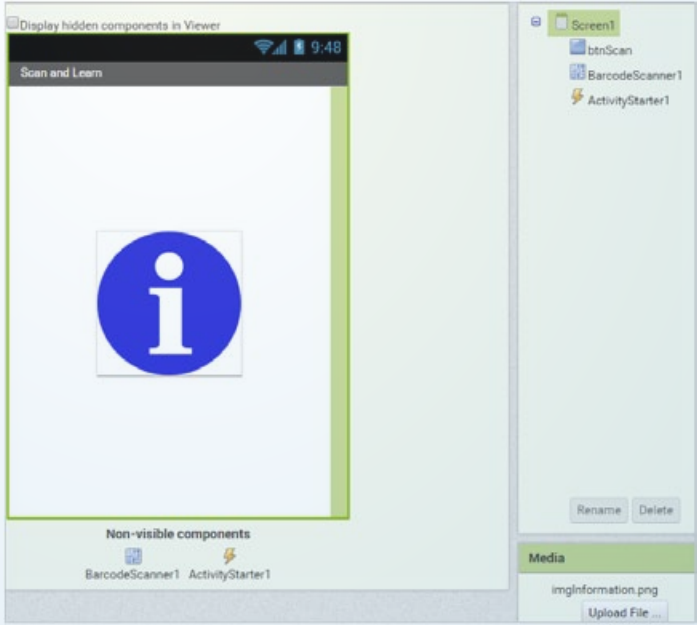


In the Resources:

“imgNaReaction.jpg” by Naatriumi_reaktsioon_veega_purustab_klaasist_anuma.jpg:
Tavoromann. Derivative work: Tony Mach - This file was derived from Naatriumi
reaktsioon veega purustab klaasist anuma.jpg. Licensed under CC BY-SA 3.0 via
Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Sodium_and_Water.png#/media/File:Sodium_and_Water.png.

“imgKReaction.jpg” by Tavoromann. Own work. Licensed under CC BY-SA 3.0 via
Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Kaalumi_reaktsioon_veega.jpg#/media/File:Kaalumi_reaktsioon_veega.jpg.

Tutorial 5: Scan and Learn

 Project name (AppName)	COL_ScanAndLearn
 App Name (Title)	Scan and Learn
 App description (AboutScreen)	<p>This app uses the BarcodeScanner and ActivityStarter components. You can press and hold to scan a barcode or QR code. The activity starter will open the content URL using the appropriate applications installed on your device.</p>
 Resources	https://goo.gl/ITOVxt
 Screenshot	



Expected Learning Outcomes

By the end of this tutorial you should be able to:

- Use the BarcodeScanner component
- Invoke external apps using the ActivityStarter component
- Rename components



Components & Attributes

Screen1

- AlignHorizontal: **Center**
- AlignVertical: **Center**
- Icon: **appicon.png**

btnScan

- Name: **btnScan**
- Image: **imgInformation.png**

BarcodeScanner1

ActivityStarter1

Blocks

```

when btnScan .LongClick
do call BarcodeScanner1 .DoScan

```

```

when BarcodeScanner1 .AfterScan
result
do call StartActivity
    result get result

```

```

to StartActivity result
do
  set ActivityStarter1 . Action to " android.intent.action.VIEW "
  set ActivityStarter1 . ActivityClass to " com.android.internal.app.ResolverActivity "
  set ActivityStarter1 . ActivityPackage to " android "
  set ActivityStarter1 . DataUri to get result
  call ActivityStarter1 .StartActivity

```

In the Resources:

“Info Simple” by Amada44. Own work. Licensed under Public Domain via Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Info_Simple.svg#/media/File:Info_Simple.svg.

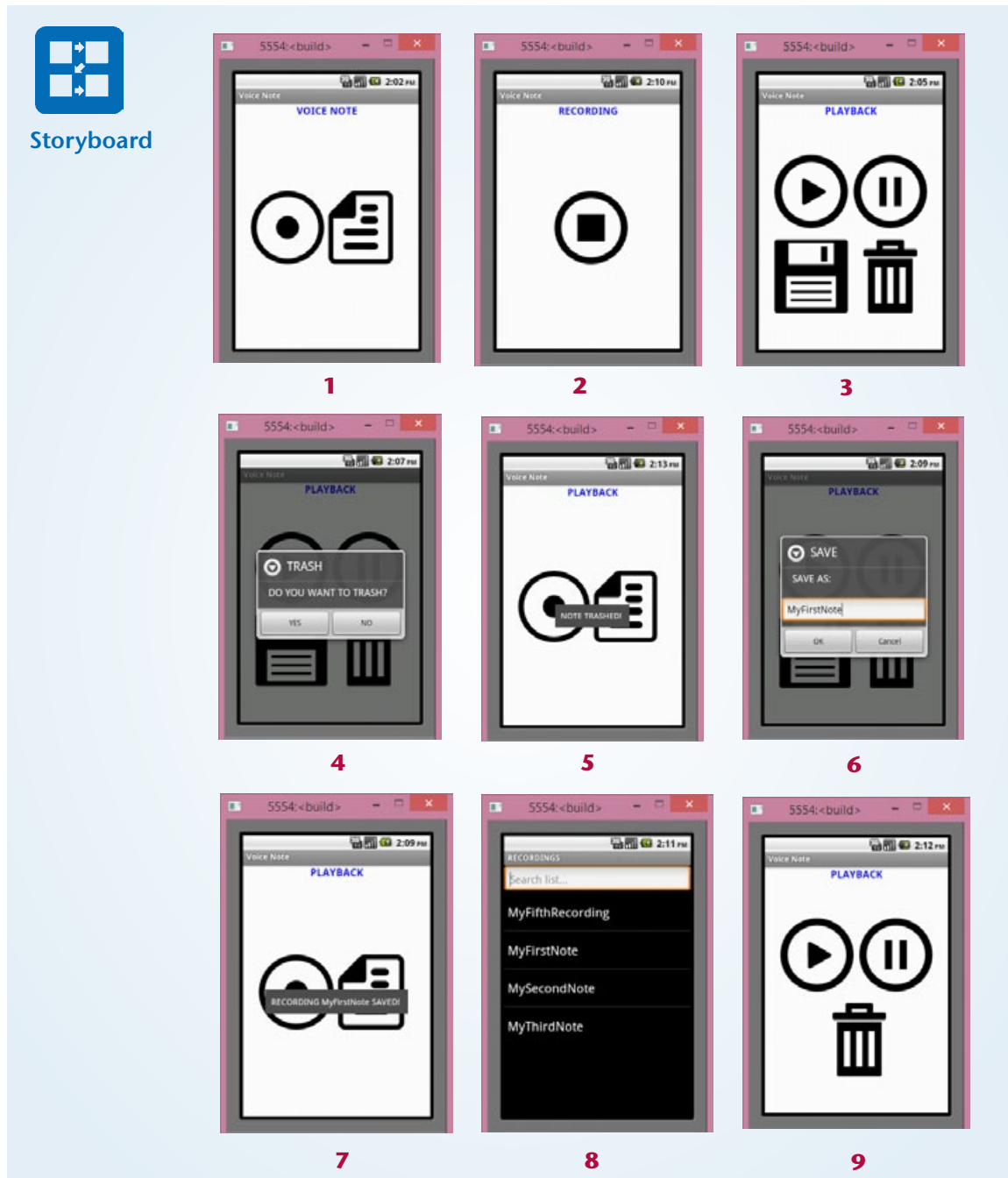
Useful Links






<https://www.the-qrcode-generator.com>.

<http://www.classtools.net/QR/create.php>.

Tutorial 6: Voice Note

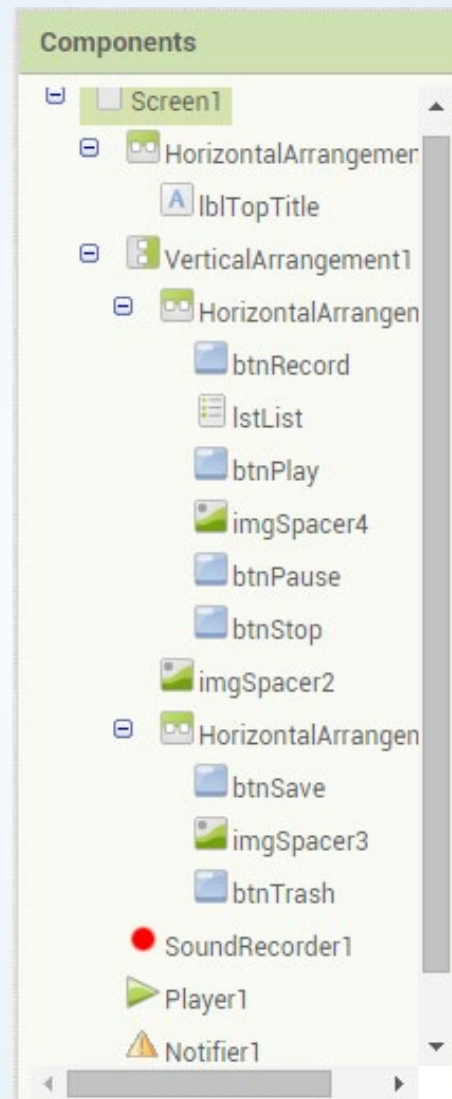
This tutorial demonstrates the concept of creating a storyboard prior to the development of an app. The idea is to design the screens one by one, following the storyboard below. The blocks are provided only as a reference. You are advised not to refer to the tutorial, for the sake of the practical hands-on work and learning. The blocks for the app should be assembled following the logic of the storyboard. Ideally, the trainer/participants would be creating the app in real-time and only using the storyboard as a guide. The storyboard will help you to structure the logical flow of work and preparation of Blocks.



	Project name (AppName)	COL_VoiceNote
	App Name (Title)	Voice Note
	App description (AboutScreen)	This voice recorder app allows students to take audio notes while they study and revise. The app uses the SoundRecorder, Player and ListPicker components to play saved notes from a list. The notes can be saved, played back or deleted from the TinyDB storage.
	Resources	https://goo.gl/6ByCkp
	Expected Learning Outcomes	By the end of this tutorial you should be able to: <ul style="list-style-type: none"> • Create a storyboard for an app • Design an app based on a storyboard

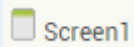


Components & Attributes

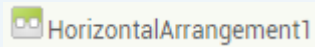




Components & Attributes



- AlignHorizontal: **Left**
- AlignVertical: **Top**
- Icon: **appicon.png**



- Width: **Fill parent**
- AlignHorizontal: **Center**



- FontBold: **True**
- FontSize: **18.0**
- Text: **VOICE NOTE**
- TextColor: **Blue**



- AlignHorizontal: **Left**
- AlignVertical: **Center**
- Width: **Fill parent**
- Height: **Fill parent**



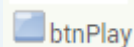
- AlignHorizontal: **Center**
- Width: **Fill parent**



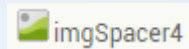
- Image: **imgRecord.png**



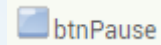
- Title: **RECORDINGS**
- Image: **imgList.png**



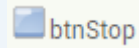
- Image: **imgPlay.png**



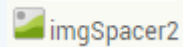
- Width: **10 pixels**



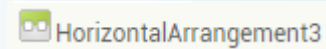
- Image: **imgPause.png**



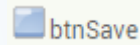
- Image: **imgStop.png**



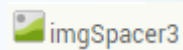
- Height: **20 pixels**



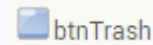
- AlignHorizontal: **Center**
- Width: **Fill parent**



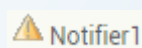
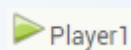
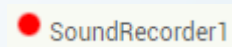
- Image: **imgSave.png**



- Width: **10 pixels**



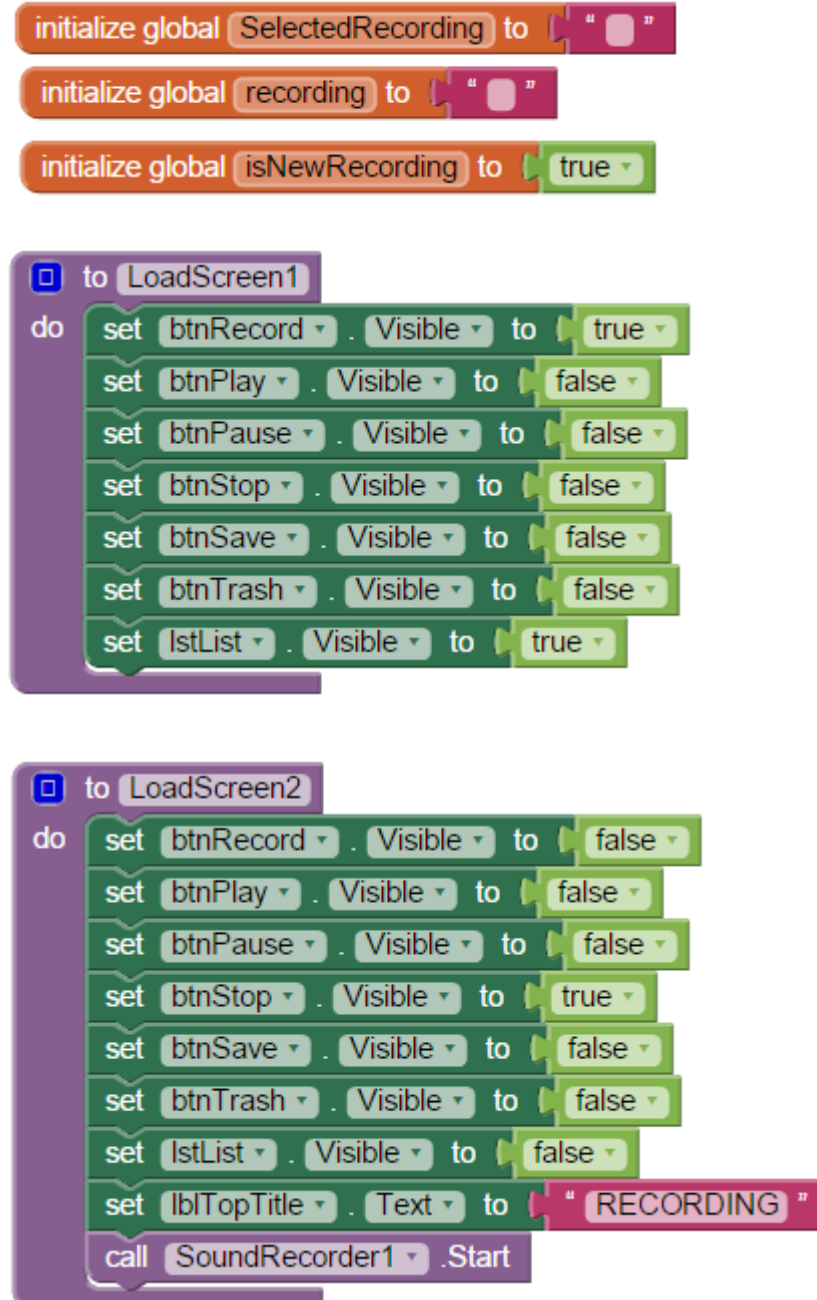
- Image: **imgTrash.png**



- NotifierLength: **Long**



Blocks



```

to LoadScreen3
do
  set btnRecord . Visible to false
  set btnPlay . Visible to true
  set btnPause . Visible to true
  set btnStop . Visible to false
  set btnSave . Visible to true
  set btnTrash . Visible to true
  set lblTopTitle . Text to "PLAYBACK"
  set lstList . Visible to false
  call SoundRecorder1 .Stop

```

```

to ReloadList
do
  set lstList . Elements to call TinyDB1 .GetTags
  set global SelectedRecording to " "

```

```

when Screen1 .Initialize
do
  call LoadScreen1
  set lstList . Elements to call TinyDB1 .GetTags

```

```

when btnRecord .Click
do
  call LoadScreen2

```

```

when SoundRecorder1 .AfterSoundRecorded
  sound
do
  set global recording to get sound
  set global isNewRecording to true

```

```

when btnStop .Click
do
  call LoadScreen3

```

```

when btnPlay .Click
do
  set Player1 . Source to get global recording
  call Player1 .Start

```

```

when btnPause .Click
do
  call Player1 .Pause

```

```

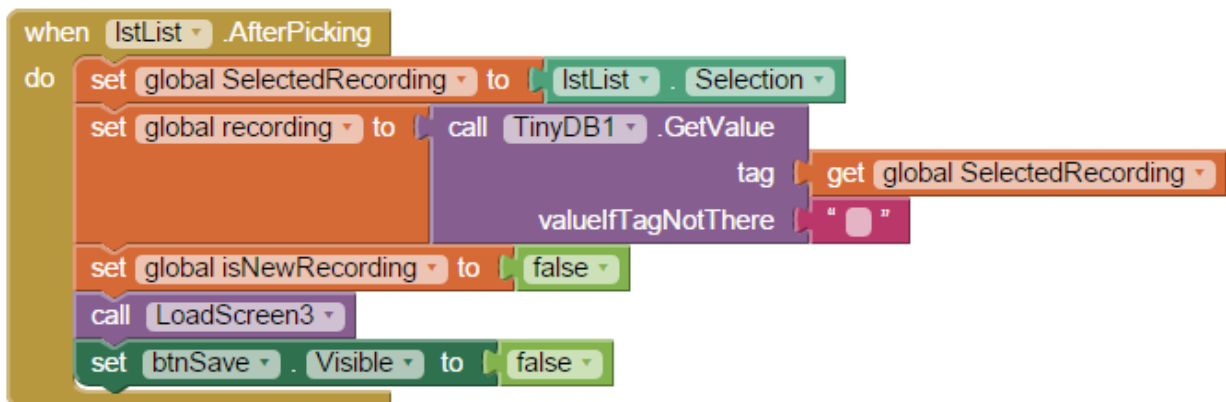
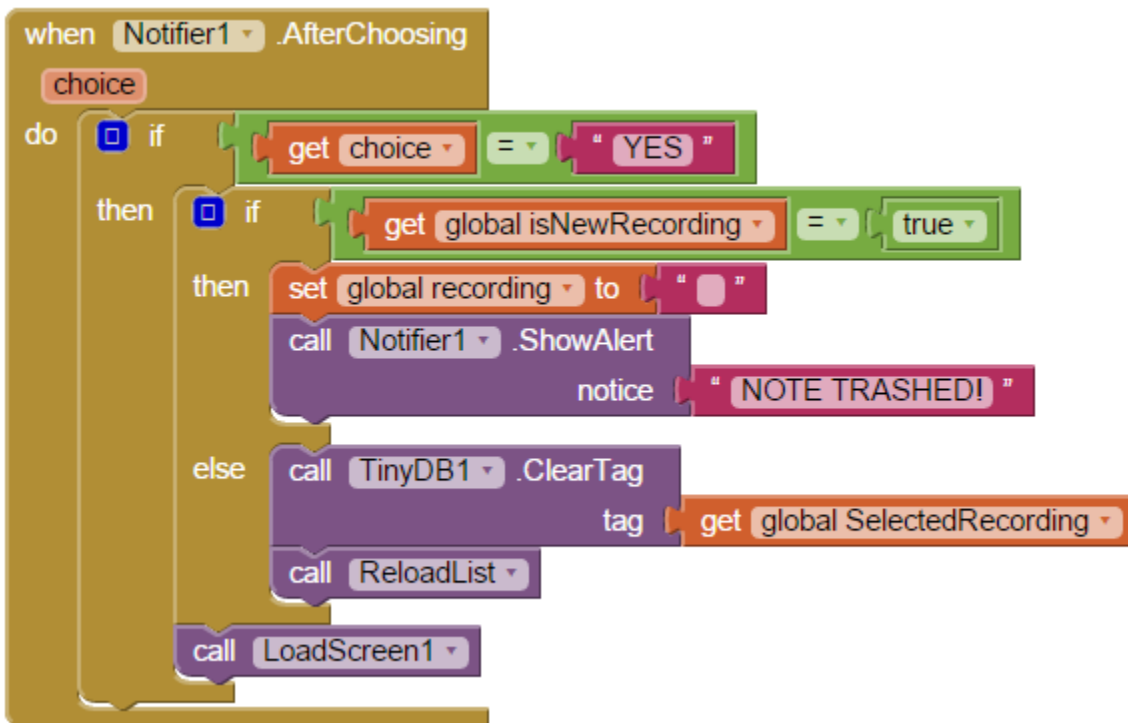
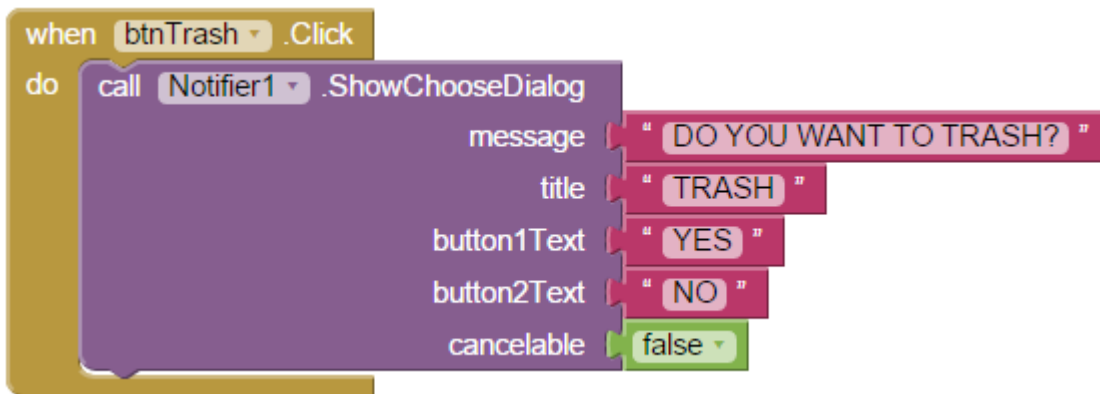
when btnSave .Click
do
  call Notifier1 .ShowTextDialog
    message "SAVE AS: "
    title "SAVE "
    cancelable true

```

```

when Notifier1 .AfterTextInput
  response
do
  if
    get response ≠ "Cancel "
  then
    call TinyDB1 .StoreValue
      tag get response
      valueToStore get global recording
    call Notifier1 .ShowAlert
      notice join "RECORDING "
        get response
        "SAVED! "
    call LoadScreen1
    call ReloadList

```



In the Resources:

“imgList.png” is based on the icon at www.freepik.com, used under a CC BY 3.0 licence from www.flaticon.com.

“imgPause.png” is based on the icon at www.google.com, used under a CC BY 3.0 licence from www.flaticon.com.

“imgPlay.png” is based on the icon at <http://appzgear.com>, used under a CC BY 3.0 licence from www.flaticon.com.

“imgStop.png” is based on the icon at www.icomoon.io used under a CC BY 3.0 licence from www.flaticon.com.

“imgRecord.png” is based on the icon at www.alessioatzeni.com, used under a CC BY 3.0 licence from www.flaticon.com.

“imgSave.png” is based on the icon at www.freepik.com, used under a CC BY 3.0 licence from www.flaticon.com.

“imgTrash.png” is based on the icon at www.freepik.com, used under a CC BY 3.0 licence from www.flaticon.com.

4. Packaging and Distribution¹

You can share your app in an **executable** form (.apk) that can be installed on a device, or in **source** code form (.aia) that can be loaded into App Inventor and remixed. You can also distribute your app on the Google Play Store.

4.1 Sharing your app so that others can remix it (.aia file)

Make sure you are viewing the list of all of your projects (if you are not, choose **Project > My Projects**). Select the project you wish to share by checking the box next to it. Choose **Project > Export selected project (.aia) to my computer** to export the source code (blocks) for your project (Figure 4.1).

The source code is downloaded in an .aia file.

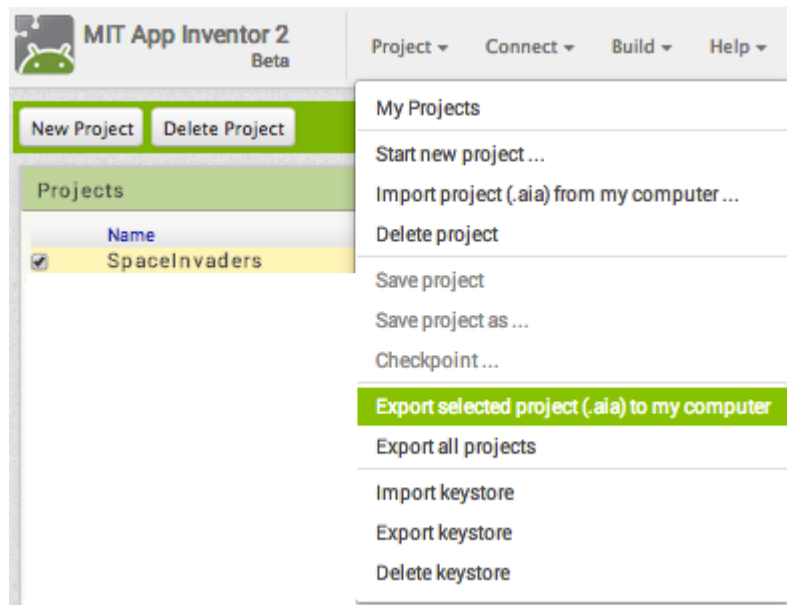


Figure 4.1 Exporting a project as an .aia file

If you send it to a friend, they can open it with **Project > Import project (.aia) from my computer** as shown in Figure 4.2.

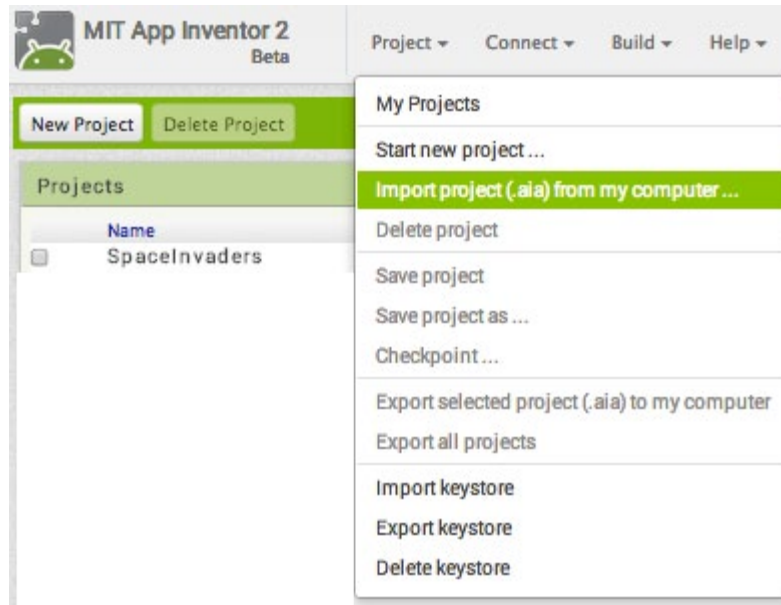


Figure 4.2 Importing a project into AI2

4.2 Sharing your app for others to install on their phone/tablet (.apk file)

Package the app (.apk file) by going to the **Build** menu on the App Inventor toolbar as shown in Figure 4.3.

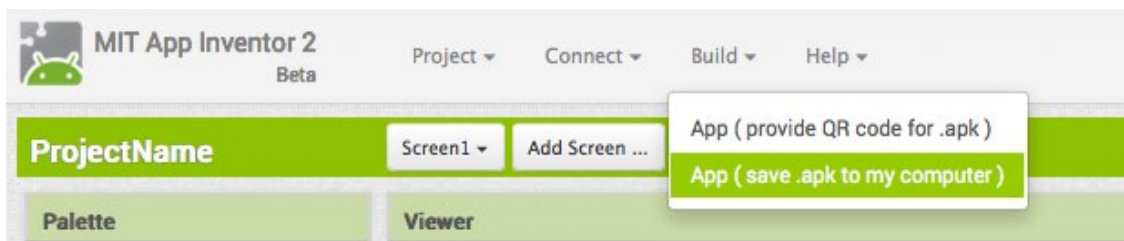


Figure 4.3 Packaging an app as a .apk file

Select **App (save .apk to my computer)**. A pop up box should alert you that your download has begun (Figure 4.4).

NOTE: The other option (provide a QR code for an .apk file) produces a scannable QR code that will download the app and make it available for two hours. You can share this code with others, but they have to use it within two hours of your generating it.



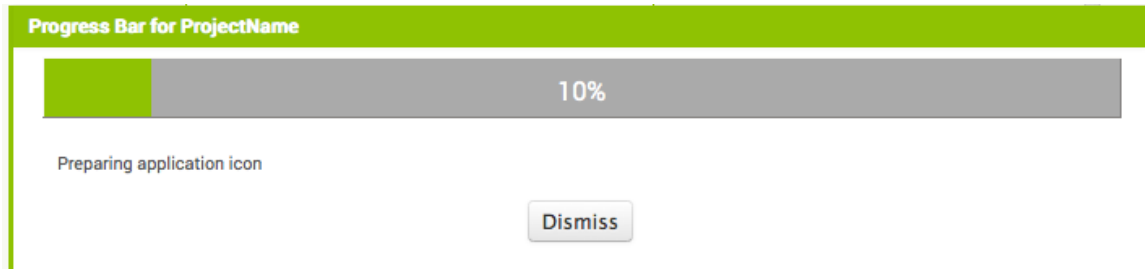


Figure 4.4 Build progress of the .apk file

Once the build completes, you can email the app (.apk file) to your friends who can install it by opening the email from their Android phones. If you want to distribute it more widely, you can upload it to a website that both you and your friends can access. You can also distribute your app via the Google Play Store.



NOTES:

1. Anyone installing your app (which is an .apk file) will need to change the setting on their phone to allow the installation of non-market applications.
2. To find this setting on versions of Android prior to 4.0, go to **Settings > Applications** and then check the box next to **Unknown Sources**. For devices running Android 4.0 or above, go to **System settings > Security** or **Settings > Security & Screen lock**, check the box next to **Unknown source** and confirm your choice.
3. The source code (.aia) files are not executable Android programs. Only .apk files are executable. The source code is also not Java SDK code — it can only be loaded into App Inventor.

5. Publishing Apps on Google Play¹

5.1 The Developer Console

Upload apps, build your product pages, configure prices and distribution, and publish. You can manage all phases of publishing on Google Play through the Developer Console from any Web browser (Figure 5.1). Once you've registered and received verification by email, you can sign in to your Google Play Developer Console.

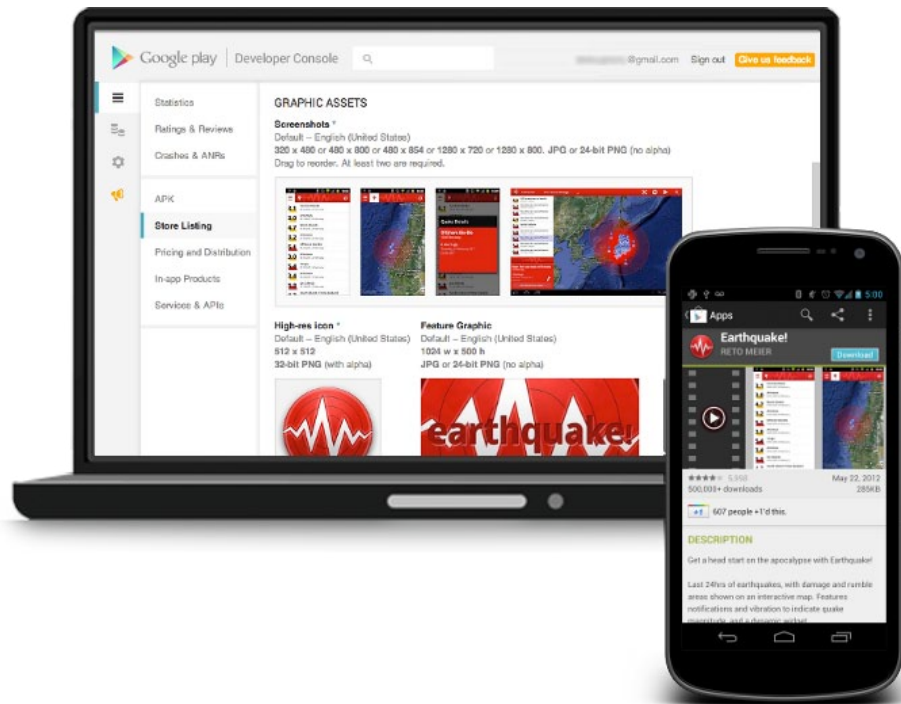
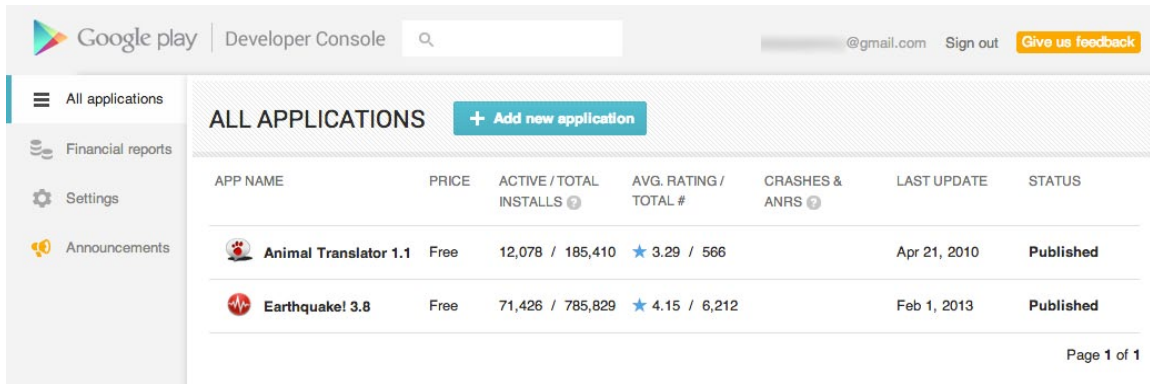


Figure 5.1 Google Play Developer Console



5.1.1 All Applications

Start in **All Applications** (Figure 5.2), which gives you a quick overview of your apps and lets you jump to stats, reviews and product details or upload a new app.

¹ Publishing Apps on Google Play is adapted from <http://developer.android.com/distribute/googleplay/developer-console.html> under a Creative Commons Attribution 2.5 Licence.



The screenshot shows the Google Play Developer Console interface. At the top, there's a header with the Google Play logo, 'Developer Console', a search bar, and user information like '@gmail.com', 'Sign out', and a 'Give us feedback' button. On the left, a sidebar menu includes 'All applications', 'Financial reports', 'Settings', and 'Announcements'. The main area is titled 'ALL APPLICATIONS' with a '+ Add new application' button. Below this is a table listing applications.

APP NAME	PRICE	ACTIVE / TOTAL INSTALLS ?	AVG. RATING / TOTAL #	CRASHES & ANRS ?	LAST UPDATE	STATUS
 Animal Translator 1.1	Free	12,078 / 185,410	★ 3.29 / 566		Apr 21, 2010	Published
 Earthquake! 3.8	Free	71,426 / 785,829	★ 4.15 / 6,212		Feb 1, 2013	Published

Page 1 of 1

Figure 5.2 All applications on the Google Play Developer Console

5.1.2 Account Details

Add basic developer profile information about yourself or your company to the **Account details** page (Figure 5.3). This identifies you to Google Play and your customers. You can go back at any time to edit the information and change your settings.

Your developer profile contains:

- Developer name: displayed on your store listing page and elsewhere on Google Play.
- Contact information: used by Google only, it isn't seen by your customers.
- Website URL: displayed on your store listing page.

On the **Account details** page you can also add restricted access for marketers and other teams, register for a merchant account, or set up test accounts for Google Play licensing.

ACCOUNT DETAILS

Saved

DEVELOPER PROFILE

Developer name *

Company or developer name...

The developer name will appear to users under the name of your application.

Email address *

your.address@gmail.com

Website

http://your.site.com

Phone Number *

+1-650-555-1212

Include plus sign, country code and area code.
For example, +1-650-253-0000.

Email updates

☐ I'd like to get occasional emails about development and Google Play opportunities.

Figure 5.3 Google Developer account details

5.1.3 Linking Your Merchant Account

If you want to sell apps or in-app products, link your Google Wallet Merchant Account to your developer profile. Google Play uses the linked merchant account for financial and tax identification, as well as for monthly payouts from sales.

5.1.4 Multiple User Accounts

Set up user accounts for other team members to access different parts of your Developer Console as shown in Figure 5.4.

INVITE A NEW USER

Email address

Choose a role for this user

Product lead ▼

- ☒ Create & edit draft apps
- ☒ Edit store listing, pricing & distribution
- ☐ Manage Production APKs
- ☒ Manage Alpha & Beta APKs
- ☒ Manage Alpha & Beta users
- ☐ Create apps distributed only to Google Apps domain
- ☐ Change distribution restriction
- ☐ View financial reports
- ☐ Reply to reviews
- ☒ Edit games
- ☐ Publish games


Figure 5.4 Multiple user account in the Developer Console

The first account registered is the account owner, with full access to all parts of the console. The owner can add user accounts and manage console access. For example, an owner can grant users access to publishing and app configuration, but not to financial reports.

5.1.5 Store Listing Details

Use the **Developer Console** to set up a **Store Listing** page (Figure 5.5). This is the home for your app in Google Play. It's the page users see on their mobile phones or on the Web to learn about your app and download it.

Upload custom brand assets, screenshots and videos to highlight what's great about your app; provide a localised description; add notes about the latest version, and more. You can update your store listing at any time.


EARTHQUAKE! – com.radioactiveyak.earthquake

Published ▾

STORE LISTING

Saved

Fields marked with * need to be filled before publishing.o

PRODUCT DETAILS

English (United States)

Add translations

Title *

English (United States)

Earthquake!

11 of 30 characters

Description *

English (United States)

Get a head start on the apocalypse with Earthquake!

Last 24hrs of earthquakes, with damage and rumble areas shown on an interactive map. Features notifications and vibration to indicate quake magnitude, and a dynamic widget.

Now optimized for tablets!

Promo text

English (United States)

Get a head start on the apocalypse! 24h of quakes mapped.

Recent changes

English (United States)


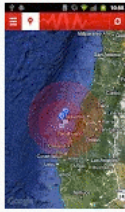


Fixed force close on refresh bug introduced in last update (sorry!)

GRAPHIC ASSETS

If you haven't added localized graphics for each language, graphics for your default language will be used.
[Learn more about graphic assets.](#)

Screenshots *

Default – English (United States)
 320 x 480 or 480 x 800 or 480 x 854 or 1280 x 720 or 1280 x 800. JPG or 24-bit PNG (no alpha)
 Drag to reorder. At least two are required.







High-res icon *

Default – English (United States)

512 x 512

32-bit PNG (with alpha)



Feature Graphic

Default – English (United States)

1024 w x 500 h

JPG or 24-bit PNG (no alpha)




Figure 5.5 Google Play Store listing details

5.1.6 Upload and Instantly Publish

From the **Developer Console** you can quickly upload and publish a release-ready Android application package file. The app is a draft until you publish it, at which time Google Play makes your store listing page and app available to users. Your app appears in the store listings within hours, not weeks.

Once your app is published, you can update it — change your prices, configuration, and distribution options, for example — as often as you want, whenever you want without needing to update your app binary.

As you add features or address code issues, you can publish an updated binary at any time. The new version is available almost immediately and existing customers are notified that an update is ready for download. Users can also accept automatic updates to your app so that your updates are delivered and installed as soon as you publish them. You can unpublish your apps at any time.

5.1.7 Alpha- and Beta- Testing

It's always valuable to get real-world feedback from users, especially before a launch. Google Play makes it easy to distribute pre-release versions of your app to alpha and beta test groups anywhere in the world.

In the **APK** section of your Google Play Developer Console you'll find **Alpha Testing** and **Beta Testing** tabs (Figure 5.6). Here you can upload versions of your apps' .apk files and define a list of testers as a Google Group or Google+ Community. You will receive a URL to forward to your testers and they can then opt in to the testing program.

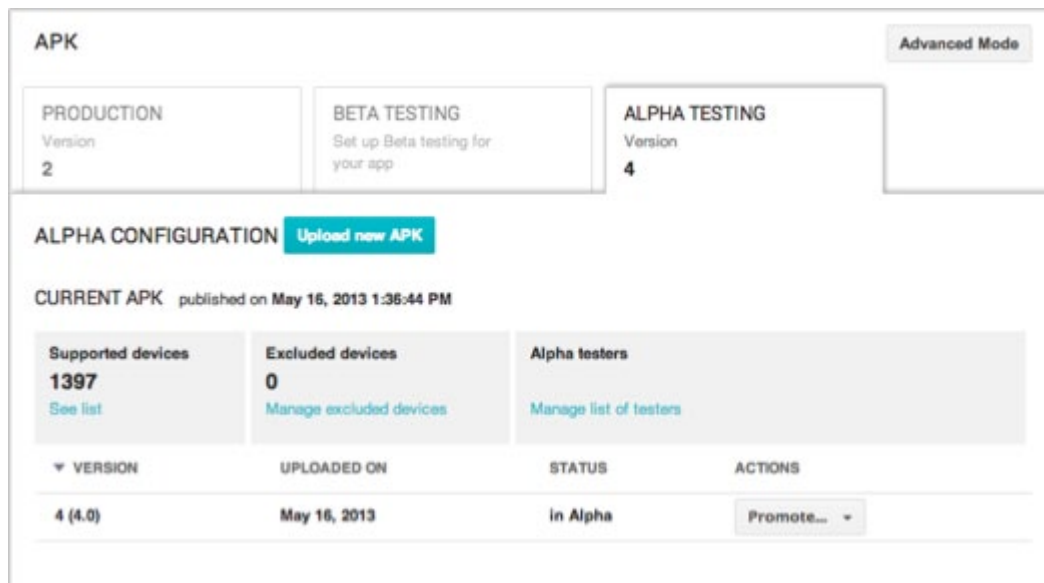


Figure 5.6 Alpha- and beta-testing your app

When your testers download the app from its product page, Google Play will deliver the alpha or beta version to them as appropriate. Incidentally, if a user happens to be opted-in to both your testing groups, Google Play will always deliver the alpha test version to them, not the beta.

Note that users cannot provide feedback and reviews on alpha and beta versions of your apps. To gather feedback, you could use the Google Group or Google+ Community, or set up an email address or your own website.

You can use these testing programs to optimise your apps, help with rollout to new markets and start building your community.

5.1.8 Staged Rollouts

You can stage the rollout of your apps using the **Production** tab in the **APK** section of your **Google Play Developer Console**. Here you can define the percentage of users who'll be able to download your app.

Staging your rollout will help limit the impact of unexpected bugs or server load and enable you to gauge user feedback with an unbiased sample of users. Users can rate and review your apps during staged rollouts, so if you're hesitant, start your rollout with a small percentage of users. Be sure to watch for and respond to any negative reviews.

Note that rollbacks aren't supported due to the app versioning requirements of the Android platform. If you need to roll back, consider launching a previous APK (app package) with a new version number. However, this practice should be used only as a last resort, as users will lose access to new features and your old app may not be forward-compatible with your server changes or data formats, so be sure to run alpha and beta tests of any updates.

5.2 Multiple APK Support

In most cases, a single APK is all you need, and it's usually the easiest way to manage and maintain the app. However, if you need to deliver a different APK to different devices, Google Play provides a way to do that.

Multiple APK support lets you create multiple app packages that use the same package name but differ in their OpenGL texture compression formats, screen-size support or Android platform versions supported. You can simply upload all the APKs under a single product listing and Google Play selects the best ones to deliver to users, based on the characteristics of their devices.

You can also upload up to two secondary downloads for each published APK, including multiple APKs, using the **APK Expansion Files** option. Each expansion file can be up to 2GB and contain any type of code or assets. Google Play hosts them for free and handles the download of the files as part of the normal app installation.

5.3 Selling and Pricing Your Products

There are tools to set prices for your apps and in-app products. Your app can be free to download or priced, with payment required before download (Figure 5.7).

<input checked="" type="checkbox"/> Singapore	SGD	1.84	incl. 0% tax
<input checked="" type="checkbox"/> Slovakia	EUR	1.08	incl. 5% tax
<input checked="" type="checkbox"/> Slovenia	EUR	1.08	incl. 5% tax
<input checked="" type="checkbox"/> South Africa			
<input checked="" type="checkbox"/> South Korea	KRW	1,608	
<input checked="" type="checkbox"/> Spain	EUR	1.13	incl. 10% tax
<input checked="" type="checkbox"/> Sri Lanka			
<input checked="" type="checkbox"/> Sweden	SEK	9.96	incl. 5% tax

Figure 5.7 Selling and pricing products on Google Play

If you publish your app as free, it must remain free for the life of the app. Free apps can be downloaded by all users in Google Play. If you publish it as priced, you can change it later to free. Priced apps can be purchased and downloaded only by users who have registered a form of payment in Google Play.

See **Supported locations for distributing applications** for a list of countries where you can distribute or sell your apps.

You can also offer in-app products and subscriptions, whether the app is free or priced. Set prices separately for priced apps, in-app products and subscriptions.

When users browse your app product pages or initiate a purchase, Google Play shows them the price they'll be charged in their local currency.

For each product, you initially set a default price in your own currency. If you do no more, Google Play will automatically set local prices once a month based on the US dollar price for your app.

However, Google Play gives you complete control over how you price your products in each country. To start, you can set fixed local prices manually from the default price, using the **Auto-convert prices now** feature. You can then review these prices and set new ones for any countries you wish — the price for each country is independent, so you can adjust one price without affecting the others. For most countries, the price you set includes taxes.

5.4 In-App Products

You can sell in-app products and subscriptions using **Google Play In-app Billing**. In-app products are one-time purchases, while subscriptions are recurring charges on a monthly or annual basis.

In the **In-app Products** section for a specific published or draft APK you:

- create product lists for in-app products and subscriptions,
- set prices, and
- publish the products with the app or withdraw obsolete products.

For details on how to implement In-app Billing, see the In-app Billing developer documentation at <https://developer.android.com/google/play/billing/index.html>. You make use of in-app products in the Premium, Freemium and Subscription monetisation models.

5.4.1 Distribution Controls

Manage the countries and territories that will distribute your apps. For some countries, you can choose which carriers you want to target. You can also see the list of devices your app is available for, based on any distribution rules declared in its manifest file.

5.4.2 Geographic Targeting

You can use controls in the Google Play Developer Console to easily manage the geographic distribution of your apps, without any changes in your application binary. You can specify which countries and territories you want to distribute to, and even which carriers to use (for some countries). When users visit the store, Google Play makes sure that they are in one of your targeted countries before downloading your app. You can change your country and carrier targeting at any time just by saving changes in the Google Play Developer Console. To help you market to users around the world, you can localise your store listing, including app details and description, promotional graphics, screenshots and more.

5.4.3 Capabilities Targeting

Google Play also lets you control distribution according to device features or capabilities that your app depends on. There are several types of dependencies that the app can define in its manifest, such as hardware features, OpenGL texture compression formats, libraries and Android platform versions, among others (Figure 5.8).

DEVICE COMPATIBILITY

Supported (2383)

All devices (2462)
✓ Supported (2383)
Unsupported (79)
Manually excluded (0)

<div>✓</div> <div>HCL ME TABLET PC U2 – M712MC</div>	<div>✓</div> <div>ASP320Q_ANDi – ASP320Q_GSM</div>
<div>✓</div> <div>Philips W632 – robot</div>	<div>✓</div> <div>Nextbook Next7D12 Tablet – M757ND</div>
<div>✓</div> <div>LePanII – LePanII_wifi</div>	<div>✓</div> <div>Philips W832 – Philips_WG-ROVER-RU_A</div>
ARCHOS Show all 12	
<div>✓</div> <div>Gamepad – A70GP</div>	<div>✓</div> <div>Archos 101 Internet Tablet – A101S</div>
<div>✓</div> <div>ARCHOS 97 CARBON – A97C</div>	<div>✓</div> <div>ARCHOS 80 COBALT – A80CO</div>
<div>✓</div> <div>ARCHOS 101G9 – A101</div>	<div>✓</div> <div>ARCHOS 97 XENON – A97XE</div>
ASUS Show all 18	
<div>✓</div> <div>EeePad Slider SL101 – SL101</div>	<div>✓</div> <div>PadFone – PadFone</div>
<div>✓</div> <div>ME301T – ME301T</div>	<div>✓</div> <div>TF201 – TF201</div>

Figure 5.8 Device compatibility

When you upload your app, Google Play reads the dependencies and sets up any necessary distribution rules. For technical information about declaring dependencies, read **Filters** on Google Play at <http://developer.android.com/google/play/filters.html>.

For pinpoint control over distribution, Google Play lets you see all of the devices on which your app is available based on its dependencies (if any). From the **Google Play Developer Console**, you can list the supported devices and even exclude specific devices, if necessary.

5.5 User Reviews and Crash Reports

The **Ratings & Reviews** section gives you access to user reviews for a specific app as shown in Figure 5.9. You can filter reviews in a number of ways to locate issues more easily and support your customers more effectively.

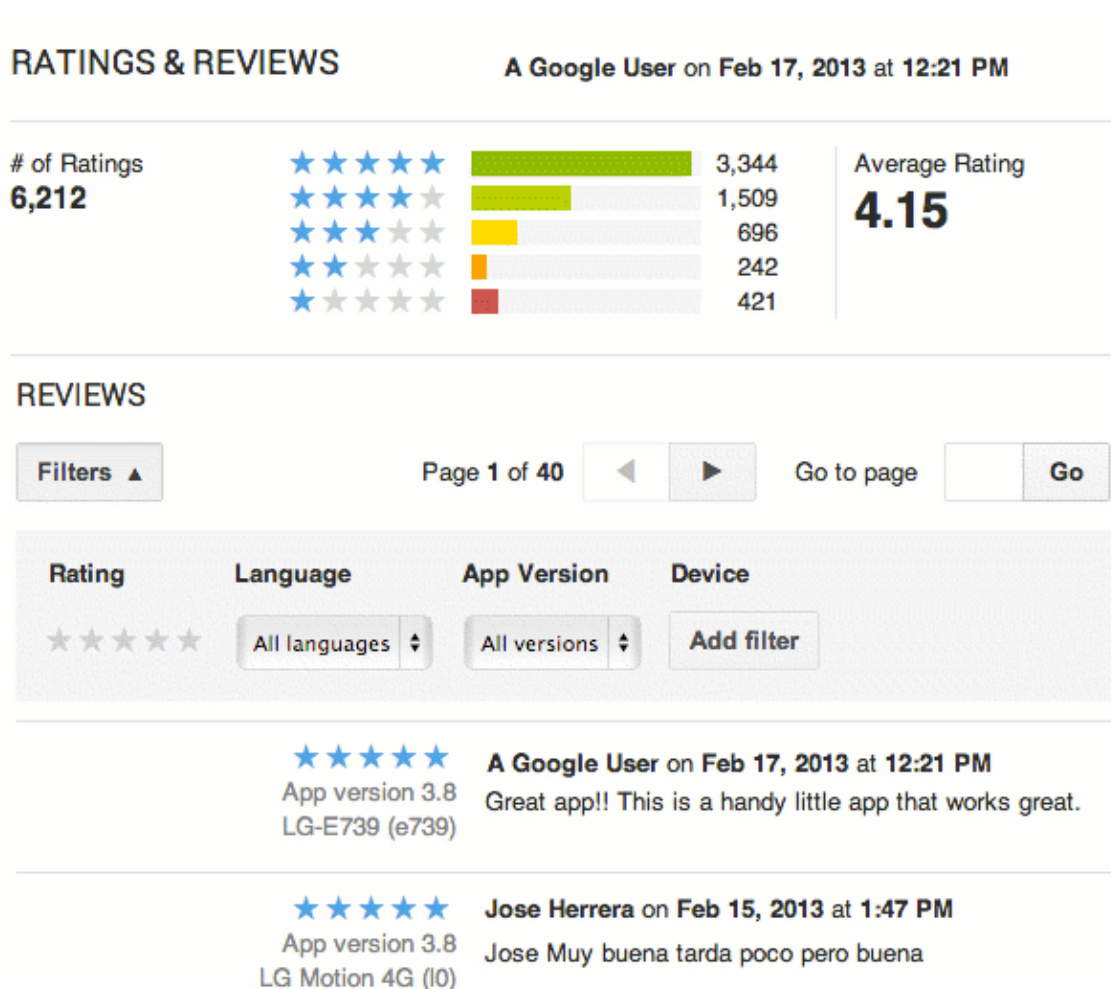


Figure 5.9 User reviews and ratings of your app

Google Play makes it easy for users to submit reviews of your app for the benefit of other users. The reviews give you usability feedback, support requests and details about important functionality issues directly from your customers.

Use crash reports for debugging and improving your app. You can see crash reports with stack trace and other data submitted automatically from Android devices.

5.5.1 App Statistics

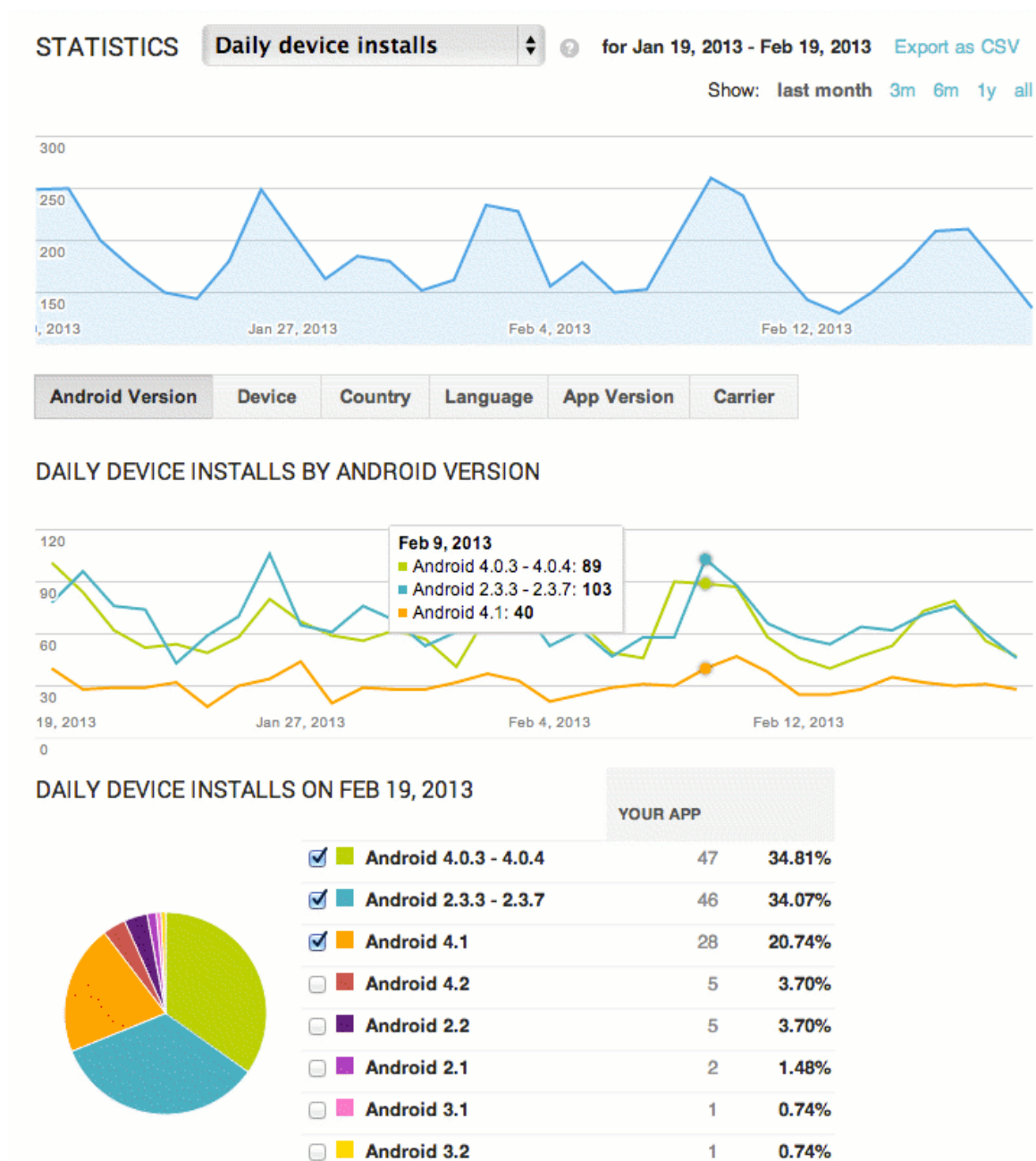


Figure 5.10 Detailed statistics about your app

The app statistics page shows you a variety of statistics about a specific app's installation performance.

6 Workshop Design and Evaluation

6.1 Workshop Training Schedule

The following is the workshop training schedule to be used when conducting the training workshop. Each session corresponds to the activities and tutorials in this toolkit.

WORKSHOP TRAINING SCHEDULE

TITLE	WORKSHOP ON EDUCATIONAL APP DEVELOPMENT FOR TEACHERS AND LEARNERS
DATES	dd to dd month year
VENUE	
TRAINER	
PREREQUISITES:	<ul style="list-style-type: none"> > Each participant must have a personal Google/Gmail account > Android smartphone or tablet (If you not have a smartphone, you can work on Emulator using a PC)

TIME	GETTING STARTED WITH ANDROID APP DEVELOPMENT <i>Day 1 (dd/mm/yyyy)</i>
8.30 – 8.45am	Registration
9.15 – 9.30am	Icebreaking
9.30 – 9.45am	Objective and expected outcome of the workshop Learning outcomes of Day 1
9.45 – 10.00am	Introduction to Visual Programming and MIT App Inventor
10.00 – 10.15am	Tea break
10.15 – 10.30am	Creating an App Inventor account
10.30 – 11.30am	Device setup for App development and debugging
11.30 – 11.45am	Introduction to the development environment (designer and blocks editor)
11.45 – 12.15pm	Introduction to available modules for App development
12.15 – 1.00pm	Using basic components: <ol style="list-style-type: none"> 1. TextToSpeech App: <i>Loud Mouth</i> 2. AccelerometerSensor App: <i>Shivers</i> 3. SpeechRecognizer App: <i>Speak to Me</i>

1.00 – 2.00pm	Lunch break
2.00 – 3.00pm	Using basic components: 4. Canvas App: <i>Scribble</i> 5. Ball App: <i>Ball Bounce</i> 6. Orientation Sensor and Clock App: <i>Move the Ball</i> 7. Camera App: <i>Say Cheese!</i> 8. Camcoder and VideoPlayer App: <i>Action Capture</i>
3.00 – 3.15pm	Tea break
3.15 – 3.45pm	Q&A Session
3.45 – 4.00pm	Recap and wrap-up

TIME	INTERMEDIATE CONCEPTS <i>Day 2 (dd/mm/yyyy)</i>
9.00 – 9.15am	Learning outcomes of Day 2
9.15 – 10.00am	Tutorial 1: Pet the Kitty
10.00 – 10.15am	Tea break
10.15 – 11.15am	Tutorial 2: Crystal Ball
11.15 – 12.30pm	Tutorial 3: Swat the Mosquito
12.30 – 1.00pm	Q&A session
1.00 – 2.00pm	Lunch break
2.00 – 3.00pm	Tutorial 4: Virtual Chemistry Experiments
3.00 – 3.15pm	Tea break
3.15 – 3.45pm	Tutorial 4: Virtual Chemistry Experiments (<i>contd...</i>)
3.45 – 4.00pm	Recap and wrap-up

TIME	ADVANCED CONCEPTS <i>Day 3 (dd/mm/yyyy)</i>
9.00 – 9.15am	Learning outcomes of Day 3
9.15 – 10.00am	Using advanced concepts: <ul style="list-style-type: none"> 1. Built-in Blocks 2. Functions 3. Variables 4. Arithmetic and Boolean Algebra 5. Control Structures 6. Exception Handling
10.00 – 10.15am	Tea break
10.15 – 11.15am	Using advanced concepts: <ul style="list-style-type: none"> 7. Starting Activities 8. Databases and Storage 9. Connectivity 10. Block Management
11.15 – 12.45pm	Tutorial 5: Scan and Learn
12.45 – 1.00pm	Packaging and distribution
1.00 – 2.00pm	Lunch Break
2.00 – 3.00pm	Storyboards and prototypes Tutorial 6: Voice Note
3.00 – 3.15pm	Tea break
3.15 – 3.45pm	Publishing Apps on Google Play
3.45 – 4.00pm	Reflections and feedback

6.2 Workshop Evaluation

The following evaluation form can be used to gather qualitative and quantitative feedback on multiple aspects of the workshop. The evaluation form is to be administered as the last session of the workshop training schedule (see section 6.1). The form has been validated and field tested. It can be administered in hard copy or online format depending on the circumstances.

EDUCATIONAL APP DEVELOPMENT FOR TEACHERS AND LEARNERS WORKSHOP EVALUATION FORM

As a participant, you are invited to evaluate this workshop. Please use this form to provide your feedback. Your personal information will be kept confidential at all times. However, your feedback might be disseminated as research findings.

PERSONAL INFORMATION

Name	<hr/>
Age	<input type="checkbox"/> 10–15 years <input type="checkbox"/> 16–20 years <input type="checkbox"/> 21–30 years <input type="checkbox"/> 31–40 years <input type="checkbox"/> 41–60 years <input type="checkbox"/> Above 60 years
Gender	<input type="checkbox"/> Male <input type="checkbox"/> Female
Occupation	<hr/>
Institution/Organisation	<hr/>
Work address	<hr/> <hr/>
Email address	<hr/>

BACKGROUND INFORMATION

Are you a teacher or a student?
(choose only one)

- ☐ Teacher
- ☐ Student

What's your IT background?
(choose only one)

- ☐ Teacher/Student (IT related subjects)
- ☐ Teacher/Student (non-IT related subjects)

What's your area of specialisation? *e.g. Education, Social Science, Mathematics, Computer Science etc.*

Do you have any prior experience in computer programming?

- ☐ No
- ☐ Yes

If yes, what's your level of expertise in computer programming?

- ☐ I have learned programming in school, university, etc.
- ☐ I am a programmer by profession
- ☐ I'm not a professional programmer but I write programs for my teaching, learning or research
- ☐ I write programs as a hobby

Do you have any prior experience in mobile application development?

- ☐ No
- ☐ Yes

If yes, please describe your experience *e.g. operating system (iOS, Android, Windows, other), programming language used etc.*

Have you worked with Visual Programming before this workshop?

- ☐ No

If yes, please describe your experience *e.g. platform, programming language etc.*

Have you used MIT App Inventor before this workshop?

- ☐ No
☐ Yes

If yes, please describe your experience *e.g. what type of applications you developed, did you publish in Google Play etc.*

Feedback on Workshop

The outcomes of the workshop are:

- Familiarise yourself with the AI2 platform
- Use the Designer and Blocks Editor
- Implement various components in applications
- Design rich user experiences (UX)
- Practise packaging and distribution of applications

What is your overall rating of this workshop?

- ☐ Poor
☐ 2
☐ 3
☐ 4
☐ Excellent

How did you feel about the length of the workshop?

- ☐ Too short
☐ Just right
☐ Too long

How did you feel about the pacing of the workshop?

- ☐ Too slow
☐ Just right
☐ Too fast

Please indicate your agreement or disagreement with the following statements using a scale of 1 to 5 where **1 = strongly disagree** and **5 = strongly agree**. Tick the appropriate box.

	STRONGLY DISAGREE				STRONGLY AGREE
	1	2	3	4	5
The objectives of the workshop have been achieved.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Your personal objectives for attending the workshop have been achieved.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Your understanding of Android app development has improved or increased as a result of the workshop.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Your skills in Android app development have improved or increased as a result of the workshop.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
You would recommend to others that they attend this workshop in the future.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The material provided was necessary for the workshop.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
There was good balance between lecture sessions, activities, tutorials and discussions.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The workshop was logically sequenced from basic to intermediate concepts.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The practical activities and tutorials were effective in teaching the concepts.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The amount of time given for the activities and tutorials was sufficient.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The amount of time given for the follow-up discussions was sufficient.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The handouts given were useful and of good quality.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There are approximately 20 lines visible. The paper has a slight shadow on the right side, suggesting it's resting on a surface.

7 Useful Resources

Icons

www.iconarchive.com

www.flaticon.com

Make Icons

<http://makeappicon.com/#>

Backgrounds

<http://subtlepatterns.com>

<http://colorcombos.com>

Logo designers

www.designmantic.com

www.graphicsprings.com

QR Code Generator

<http://goqr.me>

Connecting to MySQL

<http://puravidaapps.com/mysql.php>

Activity Starter details

<http://beta.appinventor.mit.edu/learn/reference/other/activitystarter.html>

<http://developer.android.com/distribute/googleplay/promote/linking.html#OpeningPublisher>

Embedding HTML

<http://puravidaapps.com/javascript.php>

Eclipse developer tutorial

www.vogella.com/tutorials/Android/article.html

Publishing

<http://beta.appinventor.mit.edu/learn/reference/other/appstoplay.html>

Screen Size

http://developer.android.com/guide/practices/screens_support.html

File downloads

<http://puravidaapps.com/filedownload.php>

Image Credits

All images from the <http://appinventor.mit.edu> are licensed by CC BY-SA.

Figure 1.1: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/DesignTab.png>

Figure 1.2: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/BlocksTab.png>

Figure 1.3: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/WifiA.png>

Figure 1.4: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/ai2storecompanionQR.png>

Figure 1.5: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/wifi/connectSnapshot2.png>

Figure 1.6: <http://appinventor.mit.edu/explore/sites/explore.appinventor.mit.edu/files/Six%20Character%20Code.png>

Figure 1.7: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/Emulator.png>

Figure 1.8: http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/emulator/ai_starter.png

Figure 1.9: http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/emulator/successful_ai_starter_1.png

Figure 1.10: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/emulator/connectEmulator.png>

Figure 1.11: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/emulator/connectingemulator.png>

Figure 1.12: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/emulator/middle.png>

Figure 1.13: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/USB.png>

Figure 1.14: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/ai2storecompanionQR.png>

Figure 1.15: http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/emulator/ai_starter.png

Figure 1.16: http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/emulator/successful_ai_starter_1.png

Figure 2.1: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/DesignTab.png>

Figure 2.2: <http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/BlocksTab.png>

Figures 2.3–2.45: Ishan Sudeera Abeywardena, PhD

Figure 4.1: http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/sharing_aias.png

Figure 4.2: http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/sharing_aias2.png

Figure 4.3: http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/Build_Dropdown.jpg

Figure 4.4: http://appinventor.mit.edu/explore/sites/all/files/SetupAI2/Build_Popup.jpg

Figure 5.1: <http://developer.android.com/images/gp-devconsole-home.png>

Figure 5.2: <http://developer.android.com/images/gp-dc-home.png>

Figure 5.3: <http://developer.android.com/images/gp-dc-profile.png>

Figure 5.4: <http://developer.android.com/images/gp-dc-invite.png>

Figure 5.5: <http://developer.android.com/images/gp-dc-details.png>

Figure 5.6: <http://developer.android.com/images/gp-dc-ab.png>

Figure 5.7: <http://developer.android.com/images/gp-buyer-currency.png>

Figure 5.8: <http://developer.android.com/images/gp-supported-dev-requirements.png>

Figure 5.9: <http://developer.android.com/images/gp-dc-reviews.png>

Figure 5.10: <http://developer.android.com/images/gp-dc-stats.png>



4710 Kingsway, Suite 2500
Burnaby, BC V5H 4M2
Canada

Tel: +1 604 775 8200
Fax: +1 604 775 8210
E-mail: info@col.org
Web: www.col.org

