



LIBERAL ARTS MAJORS GET IT!

# Build Android Apps with App Inventor

---

*All trademarks are acknowledged as belonging to the respective companies.*

*The Android robot is reproduced or modified from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.*

*Google Play is a trademark of Google Inc.*

*The authors thank their families and Fairleigh Dickinson University.*

*This book does not assume any programming experience whatsoever.*

---

# Table of Contents

Chapter 1. Introduction .....	1
Mobile Devices and Computers .....	1
What is an “App”? .....	2
App Inventor.....	2
App Inventor Installation.....	3
Review .....	3
Chapter and Lab Summary .....	4
Chapter 2. The Front End.....	6
Your First App .....	6
App Components .....	10
Review .....	14
Chapter and Lab Summary .....	15
Chapter 3. The Back End .....	17
Your First App .....	17
App Blocks .....	20
Review .....	21
Chapter and Lab Summary .....	21
Chapter 4. Design and Program.....	23
StartStop App.....	23
UserNamePassword App.....	26
CanvasDraw App .....	28
PickCourse App .....	32
Review .....	36
Chapter and Lab Summary .....	36
Chapter 5. Program Structure .....	38
App Perspective .....	38
Program Composition .....	38
Program Elements .....	40
Control Flow .....	41
ValidateNumber App.....	41
AssignGrade App.....	44




.....	47
Repetition With Loops .....	48
SumOfNumbers App .....	48
SumOfNumbers_While App .....	51
Review .....	54
Chapter and Lab Summary .....	55
<b>Chapter 6. Lists .....</b>	<b>57</b>
MoleMashList App .....	57
Trivia App .....	61
Review .....	65
Chapter and Lab Summary .....	65
<b>Chapter 7. Text and Colors .....</b>	<b>67</b>
String Comparisons.....	67
Joining and Splitting Strings.....	68
Extracting and Replacing Strings .....	68
Colors.....	68
Make and Split Color.....	68
Review .....	69
Chapter and Lab Summary .....	69
<b>Chapter 8. Animation and Media .....</b>	<b>71</b>
Animation .....	71
Media .....	72
TinyDB .....	72
HungryMonkey App .....	72
PaintPot_PickColorScreen App .....	77
PaintPot_Camera App .....	82
Review .....	85
Chapter and Lab Summary .....	86
<b>Chapter 9. Sensors and Connectivity .....</b>	<b>88</b>
AccelerometerSensor.....	88
BarcodeScannerSensor .....	88
Clock.....	89
LocationSensor .....	89
NearField .....	89
OrientationSensor .....	89
Connectivity .....	90
LocationMap App .....	90
StockQuote App .....	93
FindConcert App .....	94

Review .....	94
Chapter and Lab Summary .....	95
Chapter 10. Packaging Apps.....	97
Sharing the Source Code.....	97
Sharing the Executable Form.....	98
App Distribution via Google Play.....	99
Review .....	99
Index .....	100

# Chapter 1. Introduction

*“The only true wisdom is in knowing you know nothing.” –  
Socrates, Greek Philosopher.*

## ICON KEY

-  Valuable information
-  Hands-on exercise
-  Review

It's time to add the 4<sup>th</sup> **R** – **Reading**, **wRiting**, **aRithmetic** and **algoRithmic** thinking. In a world where the majority of new jobs require science, technology and math skills, it is time our Liberal Arts majors get IT (Information Technology)! While employers recognize and value the importance of liberal education and the liberal arts, they also want liberal arts graduates who are not digitally challenged. Many employers report a “skills gap” as they have trouble finding recent graduates qualified with ample digital skills to fill various positions. Meanwhile, a national educational movement in computer coding instruction is growing at lightning speeds in schools across the US and many consider coding more like a basic life skill (which might someday lead to a great job) rather than an extracurricular activity. App Inventor (AI) serves to narrow this skills gap and increase the versatility of students to become active creators of technology and “digitally” ready for the workplace rather than just being passive consumers of technology.

## Mobile Devices and Computers

Sales of hand-held devices (smartphones, tablets and phablets) are exploding. These on-line, social, and increasingly mobile computing devices are ubiquitous and offer visual, tactile and personal experiences as never before. Mobile devices in our education landscape are digital and portable - with multimedia capabilities to access the Internet, and are drastically changing the ways we teach and learn. Developing applications for such devices enables digital natives to experience mobile technology as active **creators** rather than just passive **consumers** of technology.



Computers (Fig 1.1) are electronic devices which collect and process data (input) to produce meaningful information (output). The tangible parts constitute the **hardware** and the instructions to process data into information comprise the **software**. A collection of software that controls the way the computer works and makes it possible for other programs to function is the **operating system (Systems software)**.



Fig 1.1 Computer

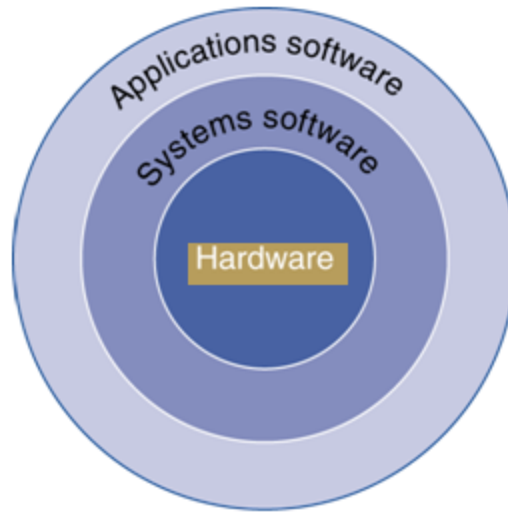


Fig 1.2 Hardware and Software

**Applications software** run on top of the underlying operating system (Fig 1.2) and cause a computer to perform useful tasks beyond the running of the computer itself.

Mobile devices are portable handheld computers with special mobile operating systems (such as Android™ or iOS). These devices allow you to do many of the same things you can do with a desktop or laptop computer.

## What is an “App”?



An “App” (or Application) is a self-contained program or piece of software designed to fulfill a particular purpose. Mobile “Apps” are the applications software which run on portable devices. Apps are developed for an underlying operating system.

Android is an operating system for mobile devices and is maintained by Google. It comes with several standard features and services such as Google Search, Google Maps, Gmail, Google Earth, etc.

## App Inventor

App Inventor (AI) (originally provided by Google and now maintained by the Massachusetts Institute of Technology) is an open source, Web-based program development tool that even beginners with no prior programming experience can use to create mobile apps (with fun and excitement). App Inventor is an intuitive visual programming environment for mobile Apps development. It runs on various operating systems and is a cloud-based tool to build apps in your web browser.

## App Inventor Installation

You can set up App Inventor and start building apps quickly. To start the setup procedure see <http://appinventor.mit.edu/explore/ai2/setup.html>. Live Testing enables you to see your app on your device as you develop it. There are three download options depending on whether you have wireless internet connection and/or an Android device (tablet, smartphone, phablet, Kindle, etc.).

**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 App Inventor Companion App for your device. First download and install the **MIT AI2 Companion App** on your phone. Next connect your computer and your device to the same WiFi network. Finally open an App Inventor project and connect it to your device.

**Option 2:** If you do not have an Android device, you'll need to install software on your computer so that you can use the on-screen Android **emulator**. The emulator works just like an Android device but appears on your computer screen instead. **We will use this option.** First you will install the **App Inventor Setup** software. Next you will launch a program named **aiStarter**, which helps the browser to communicate with the emulator. Finally you will connect to the emulator.

**Option 3:** If you do not have a wireless internet connection, you'll need to install software on your computer so that you can connect to your Android device over the Universal Service Bus (USB) interface. First install the App Inventor Setup Software. Next download and install the MIT AI2 Companion App on your phone. Finally launch the aiStarter program.

You are now ready to build apps and you will need a Gmail account to start inventing apps. The best way to get started with App Inventor is to try the beginner tutorials at <http://appinventor.mit.edu/explore/ai2/beginner-videos.html> and learn the basics.



There are two aspects to developing apps. The **Front End (AI Designer)** - How will your app look? The **Back End (AI Blocks Editor)** - How will it function? Chapters 2 and 3 elaborate further.



## Review

- Majority of new jobs require science, technology and math skills. Employers seek liberal arts graduates who are not digitally challenged. Skills gap- not enough graduates qualified with ample digital skills.
- The 4 Rs – Reading, wRiting, aRithmetic and algoRithmic thinking.



- Mobile computing devices are ubiquitous, digital and portable. They offer visual, tactile and personal experiences.
- Computers are electronic devices which process data into meaningful information.
- Software - instructions to process data into information.
- Operating system (Systems software) - controls how a computer works and enables other programs to function. Android or iOS are examples mobile operating systems.
- Applications software performs useful tasks. An App is a self-contained program for a particular purpose. Mobile Apps are the applications software which run on portable devices.
- App Inventor is an intuitive visual programming environment for mobile Apps development. It is a cloud-based tool to build apps in your web browser.
- Front End - AI Designer - app appearance.
- Back End - AI Blocks Editor - app functionality.



## Chapter and Lab Summary

Please use the space below to summarize what you learnt in this chapter along with a summary of the Labs relevant to this chapter.

[illegible]

# BUILD ANDROID APPS WITH APP INVENTOR




## CHAPTER 1

[illegible]

## Chapter 2. The Front End

*“Design can be art. Design can be aesthetics. Design is so simple, that's why it is so complicated.” – Paul Rand, an American art director and graphic designer.*

### ICON KEY

	Valuable information
	Keyboard exercise
	Review

The **App Inventor Designer** enables you to design the front end of your app, or the “**User Interface (UI)**”.



The UI enables any user to interact with mobile devices or computers. You start creating your app by first designing the UI in the Designer. Users of your app will interact with your design using the various components such as Buttons, Check Boxes, Text Boxes, etc. You design your app's UI by adding components in the Designer window.

The Designer Window has five distinct columns or panes:

- **Palette** – consists of several different tabs - User Interface, Layout, Media, Drawing and Animation, Sensors, etc. that have components which can be dragged onto the screen in the Viewer (introduced next).
- **Viewer** – contains the screen to which you add different components by dragging them from the Palette. The non-visible components are shown below the screen.
- **Components** – a hierarchical listing of all the components on your screen.
- **Properties** – displays the properties or characteristics of a component selected in the Components area.
- **Media** – lists the various media resources used by the app – for example: image, audio and video files.

### Your First App



After App Inventor has been setup you are now ready to develop your first Android app - **HelloPurr**. This app enables a user to click on a picture of a cat to hear it meow. Please see <http://appinventor.mit.edu/explore/ai2/hellopurr.html> for a detailed tutorial about this basic app. This link also allows you to download the image and audio resources you will use in this development process.

## BUILD ANDROID APPS WITH APP INVENTOR CHAPTER 2

This app uses 2 basic components (Button and Label) from the User Interface tab and a Sound component from the Media tab.

Here are the steps:

- 1) Navigate to <http://appinventor.mit.edu> in a web browser and click on the CREATE button (Fig 2.1).

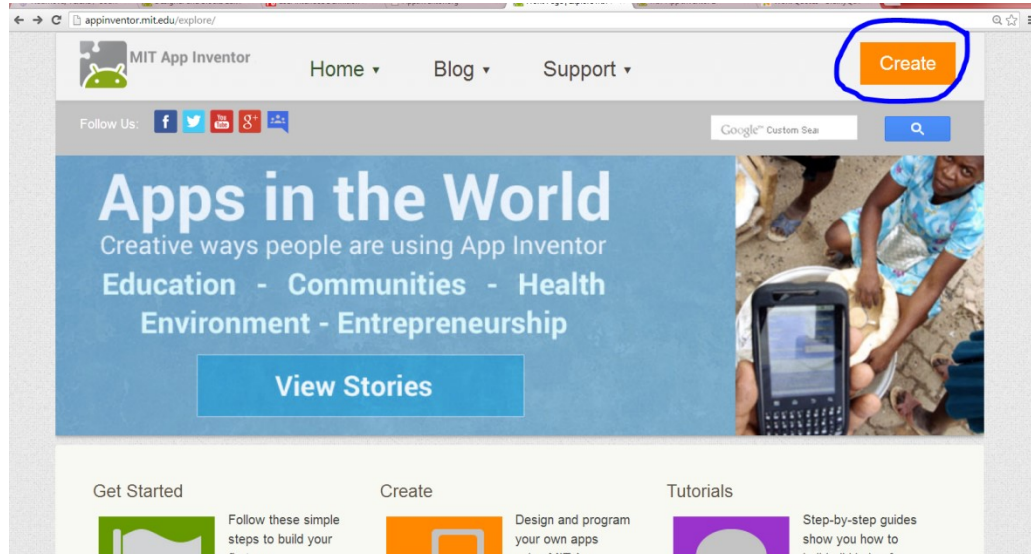


Fig 2.1 The CREATE button

- 2) Sign in using your gmail account. You may see a message about a survey – which you can take later – then you will see a welcome message click Continue and finally you will see a message that you have no projects in AI yet (Fig 2.2).

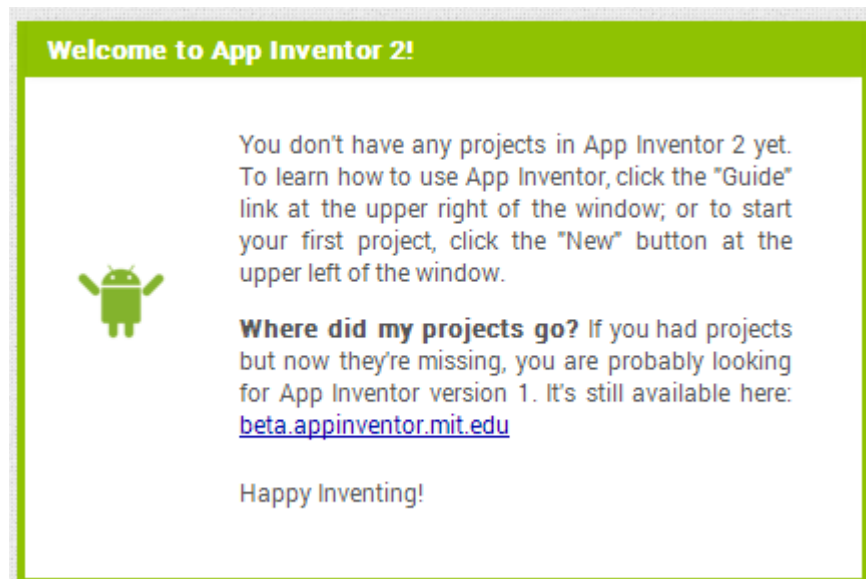


Fig 2.2 No projects

- 3) Click on New Project and create a project named HelloPurr. Now you can see the 5 distinct window panes (Fig 2.3).

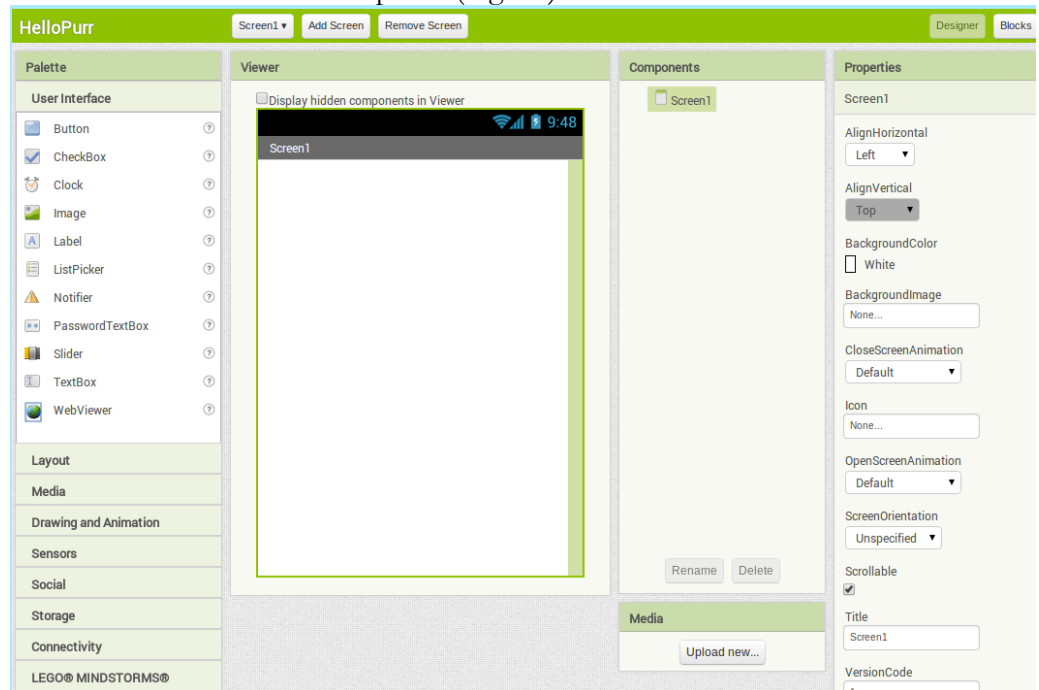


Fig 2.3 The 5 panes

- 4) You can add components from the Palette pane to the Viewer pane. First add a Button and then a Label by simply dragging them from the User Interface

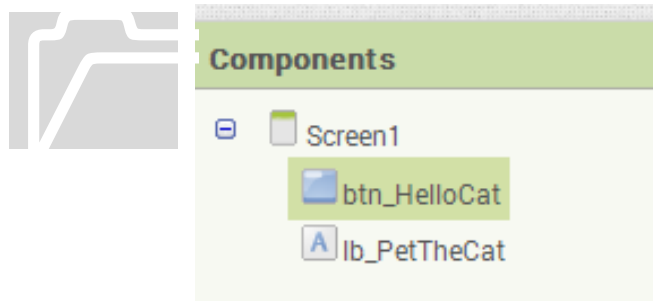


Fig 2.4 Component Names

tab of the Palette pane to Screen1 in the Viewer pane. Remember to change the names of these 2 components (Button1 and Label1) to something meaningful in the Components tab. It is a good idea to prefix the button component with **btn\_** and the label component with **lb\_**. Rename the 2 components to **btn\_HelloCat** and **lb\_PetTheCat** (Fig 2.4).

- 5) Click on a component in the Components pane to see its properties in the Properties pane. Click on the button to change two of its properties – Image and Text. Click on the Image property (None..) and click Upload File. Select the file of the cat's picture which you downloaded earlier. Now your button has the picture of the cat. Also, delete the "Text for Button1" in the Text property. Similarly, for the label, change its Text property to "***Pet the Kitty!***", change the BackgroundColor to Blue, FontSize to 30 and TextColor to Yellow (Fig 2.5).

- 6) Drag the Sound component from the Media tab of Palette pane to the Screen1 in the Viewer pane and rename it to **Snd\_Meow**. Notice the Sound component is a non-visible component and sits under the Screen1. In the Properties pane click on its Source property (None..) and click Upload File to upload the sound file (the cat's meow sound) which you saved earlier.

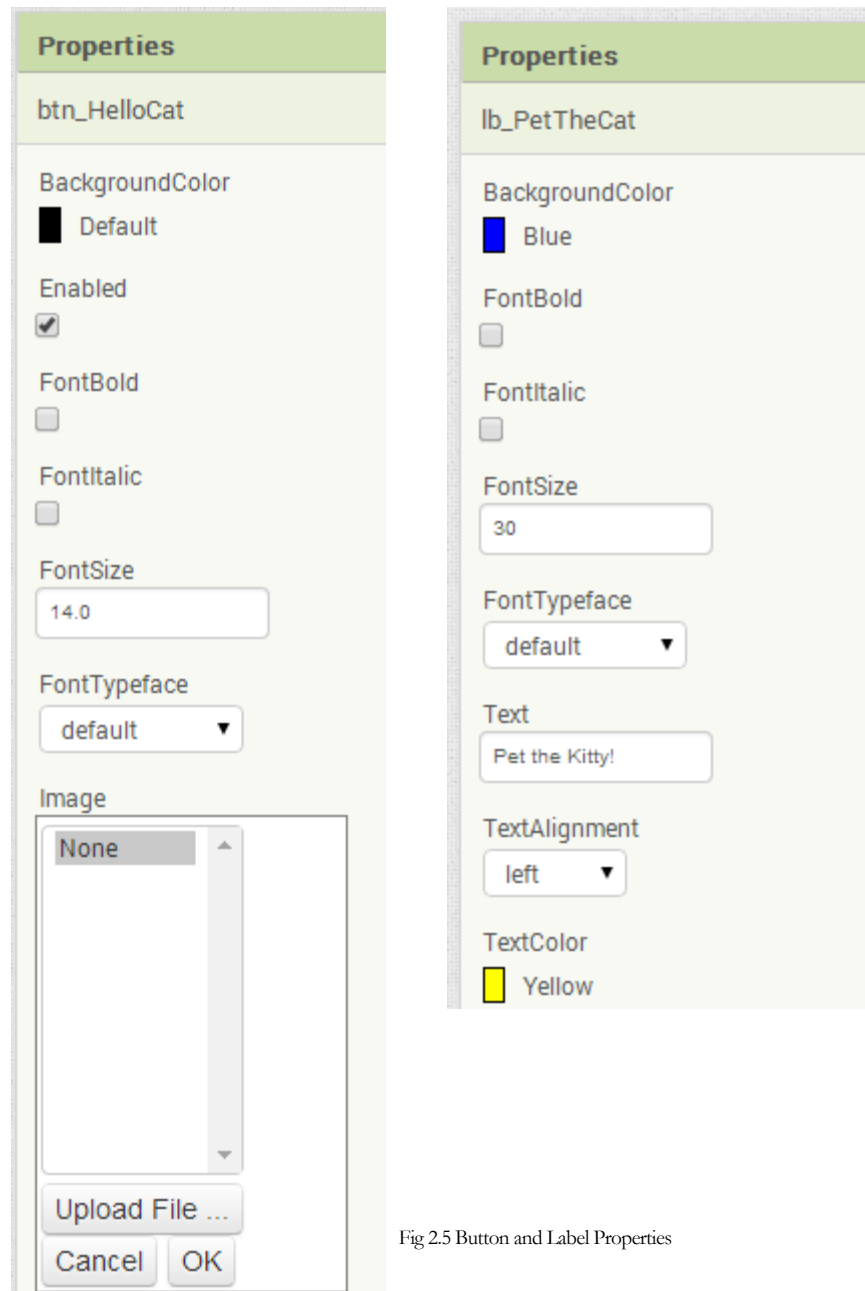


Fig 2.5 Button and Label Properties

- 7) Now the app has a picture of the cat and a sound and the screen looks like the one shown in Fig 2.6.



Fig 2.6 HelloPurr App Screen

- 8) All the above work is done in the AI Designer. Next we will add functionality to this app so that when the user pets the kitty it meows. This is the back end which will be covered in the next chapter.

## App Components

See <http://ai2.appinventor.mit.edu/reference/components/> for a detailed listing of all the components and their properties. Many components generate events which are handled in the Blocks Editor to perform some task. We look at the following tabs in the Designer's Palette pane – User Interface, Layout, Media, Drawing and Animation, Sensors, etc.

## *User Interface*

The User Interface tab contains the following components:

- **Button** – one of the easiest ways for a user to interact with an app. Clicking on a button generates a “Button Click” event, where you can add behavior (in the Blocks Editor) to do some task.
- **CheckBox** – provides a way for a user to check or uncheck an item and generates the “Checkbox Changed” event.
- **DatePicker** – used to set date and has the “DatePicker AfterDataSet” event.
- **Image** – displays images which can be set in the Designer or the Blocks Editor and has no events.
- **Label** – used to display text on the screen and has no events.
- **ListPicker** – enables a user to pick from a list and has the “ListPicker AfterPicking” event.
- **ListView** – displays a list on the screen and allows user to pick an item and has the “ListView AfterPicking” event.
- **Notifier** – displays alert messages and dialogs to the user and creates a log.
- **Password TextBox** and **TextBox** – enable users to enter text to be used by the app. **TextBox** displays the user text on the screen, whereas the **PasswordTextBox** hides the input text and displays asterisk or “\*” instead of the actual text.
- **Screen** – is a top-level component containing all other components in the app.
- **Slider** – provides a progress bar with a slider thumb which can be dragged left or right and has the “Slider PositionChanged” event.
- **Spinner** – displays a pop-up with a list of elements that can be set in the Designer or the Blocks Editor and has the “Spinner AfterSelecting” event.
- **TimePicker** – is a button which allows user to select a time when clicked and has the “TimePicker AfterTimeSet” event.
- **WebView** – enables users to view Web pages and has no events.



Many of these components are used in building apps with varying functionalities. The next chapter discusses the Back End which adds functionality and behavior to our app by using events on these components.

### *Layout*

By default, components get added one below the other on the screen in a vertical fashion. The layout tab has three arrangement components to configure the screen differently. These components serve as hidden containers to house groups of other components. They have the Height, Width and Visibility properties.

- **Horizontal Arrangement** - displays a group of components laid out from left to right.
- **Vertical Arrangement** – displays a group of components laid out from top to bottom, left-aligned.
- **Table Arrangement** – displays a group of components in a tabular fashion. In addition to height, width and visibility properties, this also has the Rows and Columns properties.

### *Media*

The Media tab has multimedia components related to images, audio, video, text, speech, player, recorder, and translation services. In addition to triggering events, most of these components also have methods which for example may start or stop recording etc.

- **Camcorder** – is used to record a video using the device's camcorder and has the AfterRecording event and the RecordVideo method.
- **Camera** - is used to take a picture using the device's camera and has the AfterPicture event and the TakePicture method.
- **ImagePicker** – a button which when tapped displays the devices image gallery and enables users to select an image. This has the AfterPicking event and the Open method.
- **Player** – plays audio and controls the vibration of the device. This has the Start, Stop, Pause and Vibrate methods.
- **Sound** – plays audio files and optionally vibrates. Needs a source which is the name of the sound file to play. This has no events but has the Play, Pause, Resume, Stop and Vibrate methods.

- **SoundRecorder** – used to record a audio and has the AfterSoundRecorded event and the Start and Stop methods.
- **SpeechRecognizer** – uses Android's speech recognition feature to convert the spoken sound into text. This has the AfterGetting event and GetText method.
- **TextToSpeech** – used to enable the device to speak text and needs the TTS Extended Service Android app to work.
- **VideoPlayer** – plays video files which can be in any of the formats - Windows Media Video (.wmv), 3GPP (.3gp), or MPEG-4 (.mp4).
- **YandexTranslate** – is powered by the Yandex Translation service and needs Internet access to translate words and sentences between different languages.

### *Drawing and Animation*

The Drawing and Animation tab has components related to movement on screen and animation using ball or image sprites (sprite – a computer graphic that can be moved around on a screen) on a touch-sensitive canvas.

- **Ball** – a round sprite placed on a canvas, can react to touch, drag, and can interact with other sprites as well as canvas edges. This has several properties (including: speed, heading and interval) and has events for detecting collisions and movement.. It also has methods to bounce, move etc.
- **Canvas** – a 2-dimensional touch-sensitive rectangular panel which is used to place and move sprites.
- **ImageSprite** – similar to a ball sprite but uses any image instead of ball.

### *Sensors*

The Sensors tab provides non-visible components that detect location, orientation and shaking of the device. Also provides a barcode scanner component and a component for some Near Field Communication (NFC).

- **AccelerometerSensor** – detects shaking and measures acceleration. This has the AccelerationChanged and Shaking events.
- **BarcodeScanner** – reads a barcode and has the AfterScan event and the DoScan method.

- **Clock** – a non-visible component - provides a timer to signal events at regular intervals and generates the “Timer” event.
- **LocationSensor** – provides location information, including longitude, latitude, altitude (if supported by the device), and address. This can also perform “geocoding” – converting an address to latitude and longitude.
- **NearField** – supports the reading and writing of text tags for NFC.
- **OrientationSensor** – determines the phone's spatial orientation and when the orientation changes, it generates the OrientationChanged event.

### *Other*

In addition to the above tabs, there are other tabs which house the Social, Storage, Connectivity etc. components.

In this chapter we looked at the design aspect of an app –The **Front End (AI Designer)** - How will your app look? In the next chapter we will explore The **Back End (AI Blocks Editor)** - How will it function?



## **Review**

- User Interface (UI) – Front End of an app – enables user to interact with mobile devices.
- AI Designer is used to design the UI. AI Designer includes Palette, Viewer, Components, Properties and Media.
- Sign in using your gmail account to create a new project.
- Build UI by dragging components from Palette to Viewer and set required properties.
- Choose App components from the following tabs: User Interface, Layout, Media, Drawing and Animation, Sensors, etc.
- User Interface tab includes basic components such as Button and Label amongst various other components.
- Layout tab has arrangement components to configure the screen.
- Media tab houses the multimedia components related to images, audio, video, text, speech, player, recorder and translation services.
- Drawing and Animation tab has components related to movement on screen and animation.
- Sensors tab provides components to detect location, orientation and shaking of device.

Please use the space below to summarize what you learnt in this chapter along with a summary of the Labs relevant to this chapter.

[illegible]

# BUILD ANDROID APPS WITH APP INVENTOR




## CHAPTER 2

[illegible]

## Chapter 3. The Back End

*“Nothing will work unless you do.” – Maya Angelou, an African-American author, poet, dancer, actress, and singer.*

### ICON KEY

-  Valuable information
-  Keyboard exercise
-  Review

The **App Inventor Blocks Editor** enables you to add functionality to your app. You can program the app's behavior simply by putting blocks together, without even writing a single line of program code.



In Chapter 2 we used the **App Inventor Designer** to add different components to the **HelloPurr** App. In this chapter we will add functionality to this app which enables a user to click on a picture of a cat to hear it meow. We are now ready to add behavior to our app. For this we use the Blocks Editor.

The Blocks Editor has two distinct columns or panes:

- **Blocks** – consists of several different tabs – Built-in, Screen1 and Any component tabs that have code blocks which can be dragged onto the Viewer (introduced next).
- **Viewer** – contains the workspace to which you add different code blocks by dragging them from the Blocks pane.

### Your First App



The **HelloPurr** app enables a user to click on a picture of a cat to hear it meow. In order to achieve this functionality we need to program the different components we have added to our screen. We do this by clicking on “**Blocks**” in the upper right corner of our **App Inventor Designer** browser above the **Properties** pane (Fig 3.1).

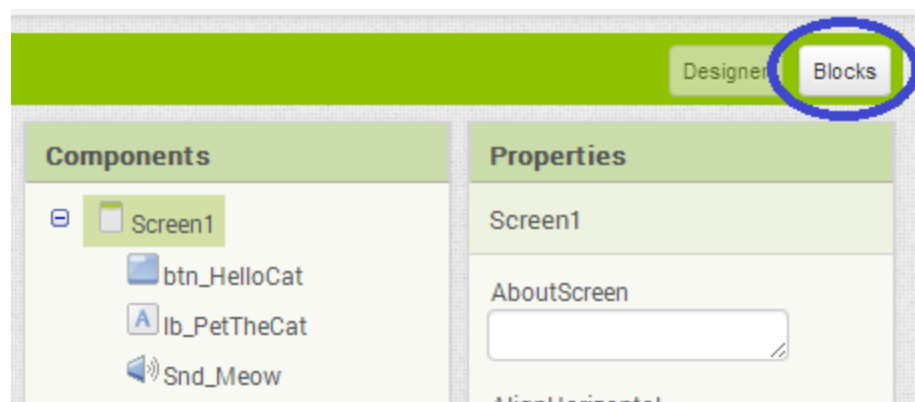
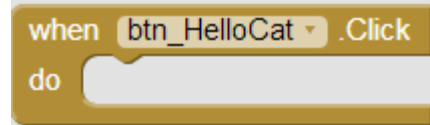


Fig 3.1 Blocks

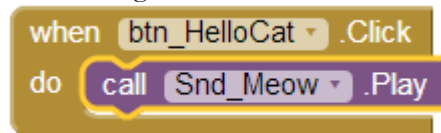
The following steps demonstrate how to achieve the required functionality.

- 1) Click on the **btn\_HelloCat** drawer in the Blocks pane of the Blocks Editor under the Screen1 tab. This opens up the events and properties associated with



this button component. Drag the event onto the workspace to the right. This is called the event handler block.

- 2) Click on the **Snd\_Meow** drawer in the Blocks pane under the Screen1 tab. This opens up the properties and commands (methods) associated with this sound component. Notice there is no event (mustard color) associated with the sound component. Drag the command block to fit into the **btn\_HelloCat**



event handler – hear a clicking sound just like a puzzle piece. The purple call piece, called the command block, is placed in the body of the event handler and is executed when the event handler runs.

- 3) Your first app is now ready to be tested on an Android device or an emulator. For now we will use the emulator. Click on **Connect** and select **Emulator**. (Fig 3.2).

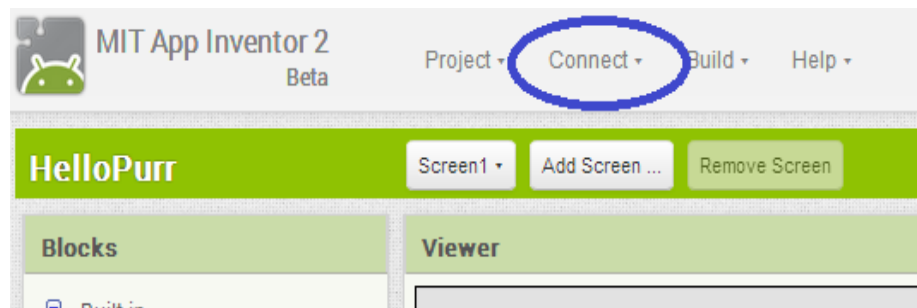


Fig 3.2 Connect to Emulator

Now you will see it trying to start the emulator (Fig 3.3).

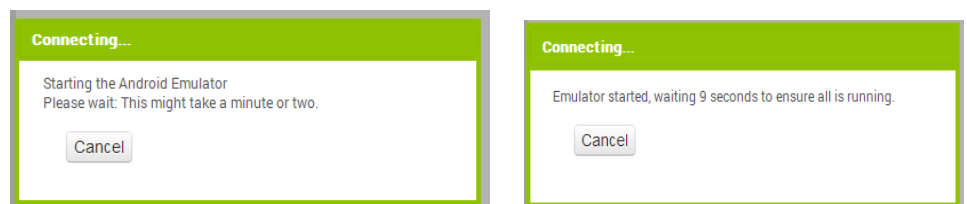


Fig 3.3 Connecting to Emulator

## BUILD ANDROID APPS WITH APP INVENTOR

### CHAPTER 3

Next you will see a black phone screen, a MIT App Inventor 2 screen and finally a Lock-Unlock screen.

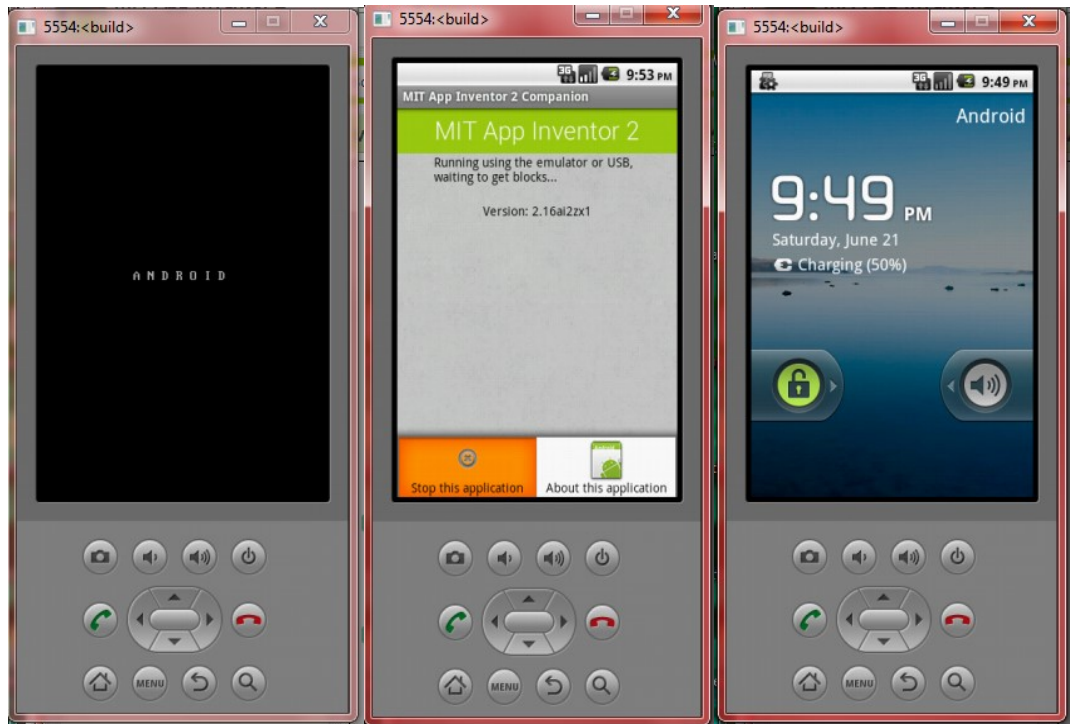


Fig 3.4 Emulator Screen



Fig 3.5 HelloPurr App

Finally (after unlocking), you will see the **HelloPurr** App on the emulator (Fig 3.5). The picture of the cat is actually the button `btn_HelloCat` which you added earlier using the AI Designer in Chapter 2. The label `lb_PetTheCat` has the message “*Pet the Kitty!*” which was set as the label’s Text property in the Designer too. The meow sound file was uploaded as the source file in the properties of sound `Snd_Meow` component. You can click anywhere on the picture of the cat to hear it meow. Try it now! Works just like it would on a real Android device.

Please note that sometimes there may be an issue with the Sound component on some devices. If you see an “OS Error” and the sound does not play - or is delayed in playing, go back into the Designer and try using a Player component (found under Media) instead of the Sound component.



## App Blocks

See <http://ai2.appinventor.mit.edu/reference/> for a detailed listing of all the blocks you can use to build your apps. We look at the following tabs in the Block editor's Blocks pane. – Built-in, Screen1, and Any component..

### *Built-in*

The Built-in tab contains the following drawers:

- **Control** – used to test a condition and perform an action based on the result, repeats an action for each element in a “for each” block or while or do loops. This also houses the controls for opening/closing “another screen” etc.
- **Logic** – used for logical operations: not, or, and. Also used for comparison operations: equality, inequality. Used to set components to constant logical values of true/false.
- **Math** – houses the various mathematical operations: <, <=, >, >=, +, -, \*, /, and mathematical functions such as random integer, random fraction, sqrt, min, max, etc.
- **Text** – houses the various string operations: join, string, split, uppercase, lowercase, replace, length, trim, segment, compare texts, contains, etc.
- **List** – houses the various list operations: join, create, make list, add items to list, is list empty, is item in list, etc.
- **Colors** – houses the various colors and operations: make color, split color, etc.
- **Variables** – houses the various operations to create, set and get variables which will be used in your code blocks.
- **Procedures** – houses the various procedure (sequence of blocks together into a group) operations.

### *Screen1*

The Screen1 tab contains a drawer for each of the components you have added in the Designer. For the **HelloPurr** App you will find the button, the label, and the sound components under Screen1 tab. You can drag any event handlers and command blocks to use in your code blocks in your workspace as we did in the **HelloPurr** App when we wanted the sound to play on the click of the button.

### *Any component*

The Any component tab is an advanced feature and contains the drawers that allow you to get/set different properties and call functions on any component. This may not mean much for a small app but when you have bigger apps with hundreds of blocks in place and you do not want to repeat the same thing multiple times, you will really appreciate this convenient feature. For example, if you have a App with two labels, you can set the label background color for either label by using this feature.

In this chapter we looked at the functional or program aspect of an app – The **Back End (AI Blocks Editor)** - How will it function? In the next chapter we will explore the combination of the design and program aspect.



## Review

- App Inventor Blocks Editor – Back End of an app – enables you to add functionality to your app.
- Program your app's behavior by putting blocks together.
- The Blocks editor has Blocks and Viewer panes.
- Blocks pane contains – Built-in, Screen1 and Any component tabs that have code blocks which can be dragged onto the Viewer.
- Viewer contains the workspace to which you add code blocks by dragging them from the Blocks pane.
- The Built-in tabs have different drawers for different functionality such as Control, Logic, Math, Text, etc.
- Your app can be tested on an Android device (smartphone or tablet) or an emulator.



## Chapter and Lab Summary

Please use the space below to summarize what you learnt in this chapter along with a summary of the Labs relevant to this chapter.

Chapter Summary

# BUILD ANDROID APPS WITH APP INVENTOR




## CHAPTER 3

[illegible][illegible]

## Chapter 4. Design and Program

*“The most important property of a program is whether it accomplishes the intention of its user.” – C. A. R. Hoare, a British Computer Scientist and a fundamental contributor to the definition and design of programming languages.*

### ICON KEY

-  Valuable information
-  Keyboard exercise
-  Review

The previous two chapters introduced you to various design components and demonstrated how to add behavior to your app simply by putting blocks together, without even writing a single line of program code.



In this chapter we will focus on how to use the different components and the program code blocks to generate something meaningful. An important requirement of an app is that it should accomplish what the user wants.

### StartStop App

Design and program an app which has two buttons – Start and Stop and displays appropriate status and image when a button is pressed. Buttons are useful as they can trigger some action when clicked. We will use the following components in this app.

#### Components

- **Label** – to display the status and for blank lines on screen
- **Button** – two buttons for Start and Stop
- **Image** – two images for Start and Stop
- **Horizontal Arrangement** – to hold buttons and images



Start a **New Project** named **StartStop**. The AI Designer will open. Set the Screen1's **Title** property to “StartStop”. You may also fill in the **AboutScreen** property to “App for Start Stop” – this will be displayed when user clicks on **About this Application** when your app runs.



The screen also has an **Icon** property where you can upload an icon image file, not more than 150 x 150 pixels (picture element) in height and width, which will display on the device instead of the default App Inventor image. This does not matter when we test using an emulator.

Using the AI Designer add the following components to Screen1 and change some of the properties.

## *Designer*

- Label **lb\_Status** (with Properties: BackgroundColor - Yellow, FontBold, FontSize - 18, Text - Current Status -, Width - Fill parent)
- Label **lb\_Blank** (with Properties: FontSize - 36, empty Text)
- Horizontal Arrangement **HA\_Images** (with default Properties) with
  - Label **lb\_LeftSpace** (with Properties: Text - 30 blank characters)
  - Image **im\_Start** (with Properties: Picture - start.png, Visible: hidden)
  - Image **im\_Stop** (with Properties: Picture - stop.png, Visible: hidden)
- Label **lb\_Blank2** (with Properties: FontSize - 36, empty Text)
- Horizontal Arrangement **HA\_Buttons** (with Properties: width - Fill parent) with
  - Button **btn\_Start** (with Properties: BackgroundColor - Green, FontBold, FontSize - 18, Text - START, Width - Fill parent). Note that buttons are enabled by default.
  - Button **btn\_Stop** (with Properties: BackgroundColor - Red, FontBold, FontSize - 18, Text - STOP, Width - Fill parent).

Fig 4.1 displays the Designer - Viewer and Components.

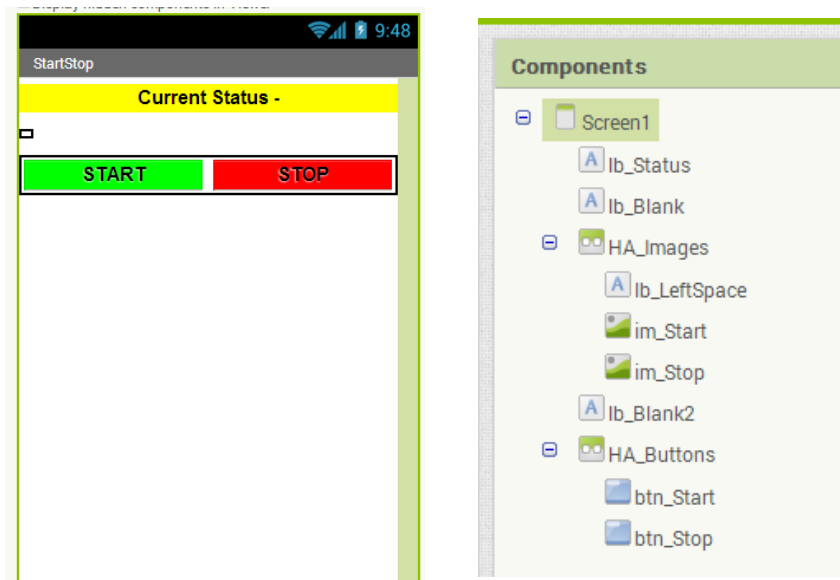


Fig 4.1 Designer - StartStopApp

Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

### *Blocks*

- Locate the **btn\_Start** in the Blocks pane and drag the click event to the



workspace. Inside this add several code blocks to do the following

- set background color to Green and text of **lb\_Status** to “Current Status – Start Button Pressed”
  - make **im\_Start** visible and **im\_Stop** invisible
  - enable **btn\_Stop** and disable **btn\_Start**
- Duplicate this code block and make necessary changes for **btn\_Stop** click event.

Fig 4.2 displays the Blocks.

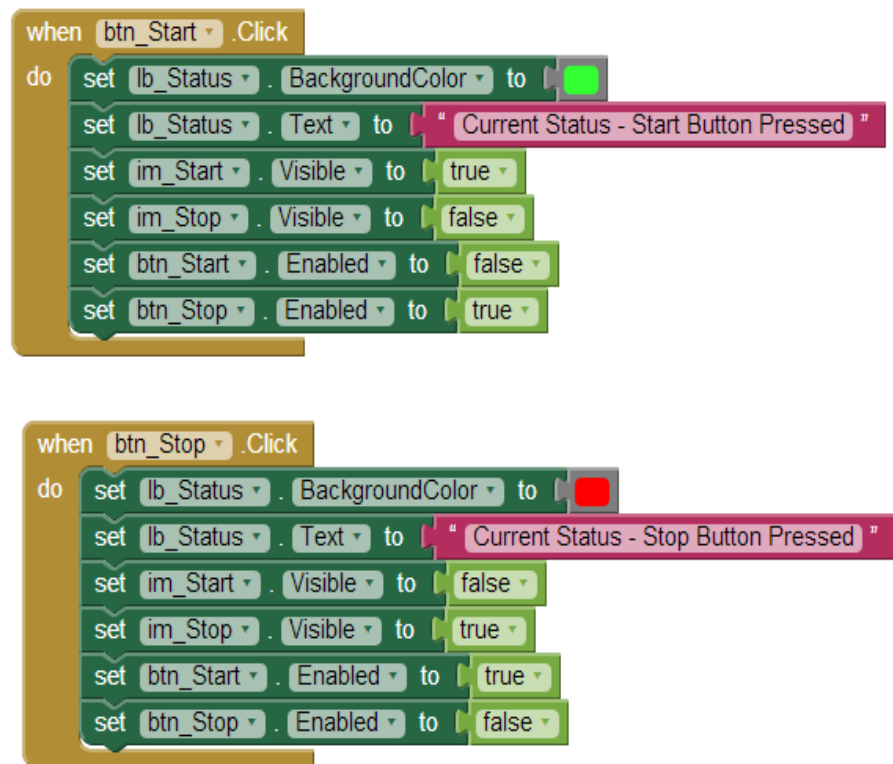


Fig 4.2 Blocks - StartStop App

## UserNamePassword App

Design and program an app which lets a user enter UserName in a textbox and Password in a password textbox and has a button – Display which displays the username on to the screen when pressed.



A **TextBox** component enables users to input text (which is visible on the screen) into your app. A **PasswordTextBox** also allows users to input text but this text is displayed with “\*” on the screen and useful for allowing users to input *sensitive* data such as passwords etc. We will use the following components.

### Components

- **Label** – to display messages on screen
- **TextBox** – to enable user to enter UserName
- **PasswordTextBox** – to enable user to enter password
- **Button** – a buttons for Display
- **Horizontal Arrangement** – to hold labels, textbox, password textbox etc.



Start a **New Project** named **UserNamePassowrd**. The AI Designer will open. Set the Screen1's **Title** property to “UserNamePassword”. You may also fill in the **AboutScreen** property to “App for UserName and Password” – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

### Designer

- Horizontal Arrangement **HA\_UserName** (set Width property to Fill parent ...) containing
  - Label **lb\_EnterUserName** (with Properties: BackgroundColor - Cyan, Text – Enter User Name;, Width – Fill parent)
  - TextBox **tb\_UserName** (with Properties: Hint – Enter your UserName, Width – Fill parent)



The TextBox has a **Hint** property where you can enter a hint for the user . This is helpful so that the user knows what your app expects in that textbox.

- Horizontal Arrangement **HA\_Password** (set Width property to Fill parent ...) containing
  - Label **lb\_EnterPassword** (with Properties: BackgroundColor – Pink, TextColor - Blue, Text – Enter Password;, Width – Fill parent)

- PasswordTextBox **ptb\_Password** (with Properties: Hint – Enter your Password, Width – Fill parent)



When a user types in a **PasswordTextBox**, its contents simply show up as a string of \*. This is very useful when you are requesting sensitive data from users (such as passwords, etc.).

- Button **btn\_Display** (with Properties: BackgroundColor – Blue, Shape – Oval, Text – Display, TextAlignment – Center, TextColor – Yellow, Width – Fill parent).
- Label **lb\_Display** (with Properties: BackgroundColor - Yellow, Text – Your UserName, TextColor – Blue, Width – Fill parent)

Fig 4.3 displays the Designer - Viewer and Components.

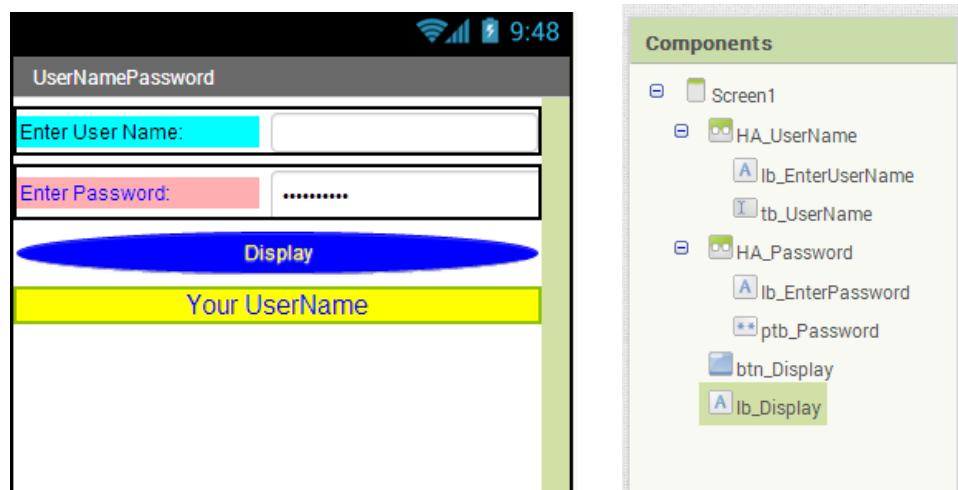


Fig 4.3 Designer - UserNamrPassword App

Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

## *Blocks*

- Locate the **btn\_Display** in the Blocks pane and drag the click event to the workspace. Inside this add several code blocks to do the following

- set **lb\_Display**.Text to “Your Username is: “ joined with **tb\_UserName**.Text. For this we use the *join* code block



found in the Built-in Text block, which simply joins two strings. The blue and white box is a *mutator* that allows blocks to expand, shrink, or even change functionality.

- clear out the text in **tb\_UserName** and **ptb\_Password**.



Fig 4.4 displays the Blocks.

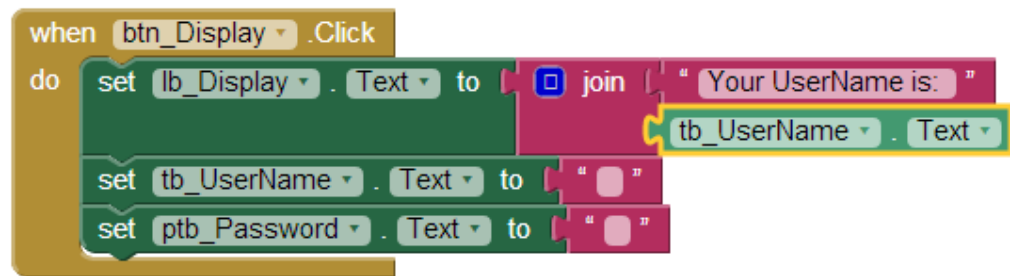


Fig 4.4 Blocks - UserNamePassword App

## CanvasDraw App

Design and program an app which uses a Draw button to draw shapes and text on a canvas. Two buttons Draw and Clear are used for drawing and clearing.



### Components

A **Canvas** component allows users to draw shapes and text. It also provides a touch sensitive area which we will explore in the chapter on Animation. For this app, we will use the following components.

- **Canvas** – to draw circle and lines and write text
- **Buttons** – two buttons for Draw and Clear
- **Horizontal Arrangement** – to hold buttons



Start a **New Project** named **CanvasDraw**. The AI Designer will open. Set the Screen1's **Title** property to "CanvasDraw". You may also fill in the **AboutScreen** property to "App for CanvasDraw" – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

### Designer

- Canvas **cn\_Blank** (with Properties: BackgroundColor - LightGrey, PaintColor – Red, Width – 300 pixels and Height - 300 pixels)



The Canvas has a **BackgroundImage** property where you can upload an image file. We will not set this property in the Designer. Instead, we will first upload an image file (chakra.png) in the Media pane and then use the Screen initialize code block to set the BackgroundImage to the image contained in this file.

- Horizontal Arrangement **HA\_DrawClear** (set Width property to 300 pixels) containing

- Button **btn\_Draw** (with Properties: BackgroundColor – Green, Text – Draw, Width – Fill parent)
- Button **btn\_Clear** (with Properties: BackgroundColor – Orange, Text – Clear, Width – Fill parent)

Fig 4.5 displays the Designer - Viewer and Components.

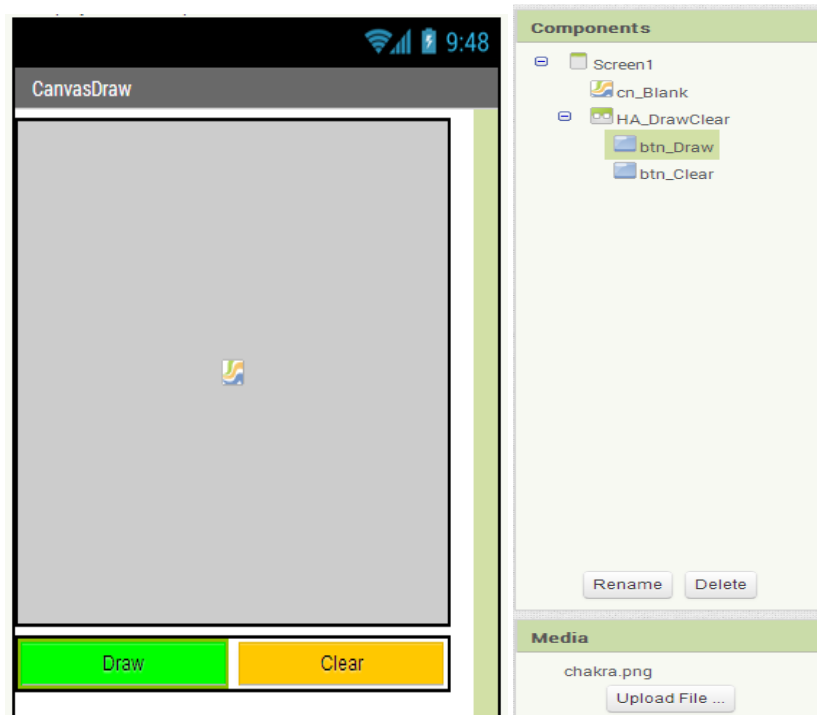


Fig 4.5 Designer - CanvasDraw App

Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

### *Blocks*


- Locate Screen1 in the Blocks pane and drag the when screen1 Initialize event to the workspace. Inside this add a code block to do the following
  - set **cn\_Blank.BackgroundImage** to “chakra.png“ which had been previously uploaded in the Media pane, taking care to match the filename exactly to the file name in the Media pane of the designer.
- Locate the **btn\_Draw** in the Blocks pane and drag the click event to the workspace. Inside this add several code blocks to do the following

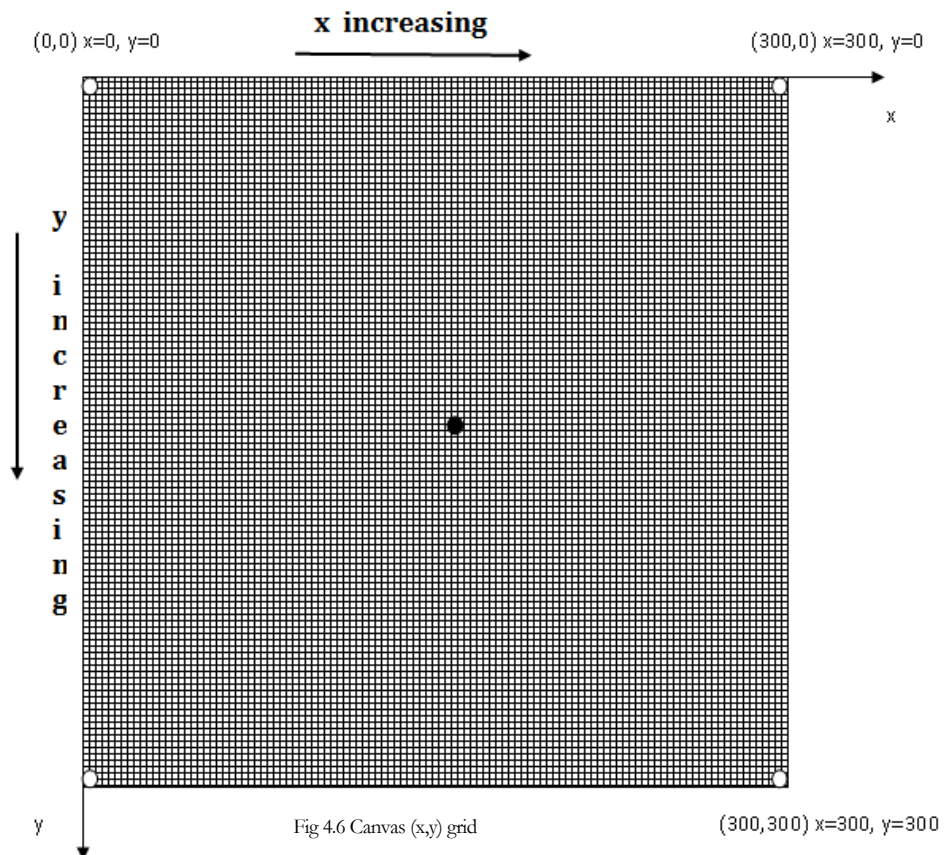
- set **cn\_Blank.PaintColor** to .

- draw a circle using **cn\_Blank.DrawCircle** method (with  $x = 150$ ,  $y = 150$  and  $r = 150$ ). This method Draws a circle (filled in) at the given (x, y) coordinates on the canvas, with the radius r.



The App Inventor coordinate system is based on an (x,y) grid (Fig 4.6) where x is the horizontal axis and y is the vertical axis. The upper left corner of the screen is location (0,0). i.e.  $x=0$  and  $y=0$ . The **x coordinates get larger** as you move to the **right** and the **y coordinates get larger** as you move **down**. (Please note that this is different from a mathematical coordinate grid where x increases to the right but y decreases as you move down).

- set **cn\_Blank.PaintColor** to 
- draw **2** lines using **cn\_Blank.DrawLine** method from (x1,y1) to (x2,y2) (with  $x1 = 150$ ,  $y1 = 150$ ,  $x2 = 75$ ,  $y2 = 25$ ) and from (x1,y1) to (x2,y2) (with  $x1 = 150$ ,  $y1 = 150$ ,  $x2 = 225$ ,  $y2 = 25$ ) These methods draw two black lines on the red circle from a common point (150,150).




- set **cn\_Blank.PaintColor** to 
- write text on the canvas using **cn\_Blank.DrawTextAtAngle** Draws the specified text (“Have a slice!”) starting at the specified (x,y) (75, 250) coordinates at the specified angle (15 degrees) using the values of the FontSize and TextAlignment properties as set in the designer.
- Locate the **btn\_Clear** in the Blocks pane and drag the click event to the workspace. Inside this call **cn\_Blank.Clear** method to clear the canvas. This removes all the drawings (the circle, the 2 lines and the text) we made earlier using the Draw method.

Fig 4.7 displays the Blocks.

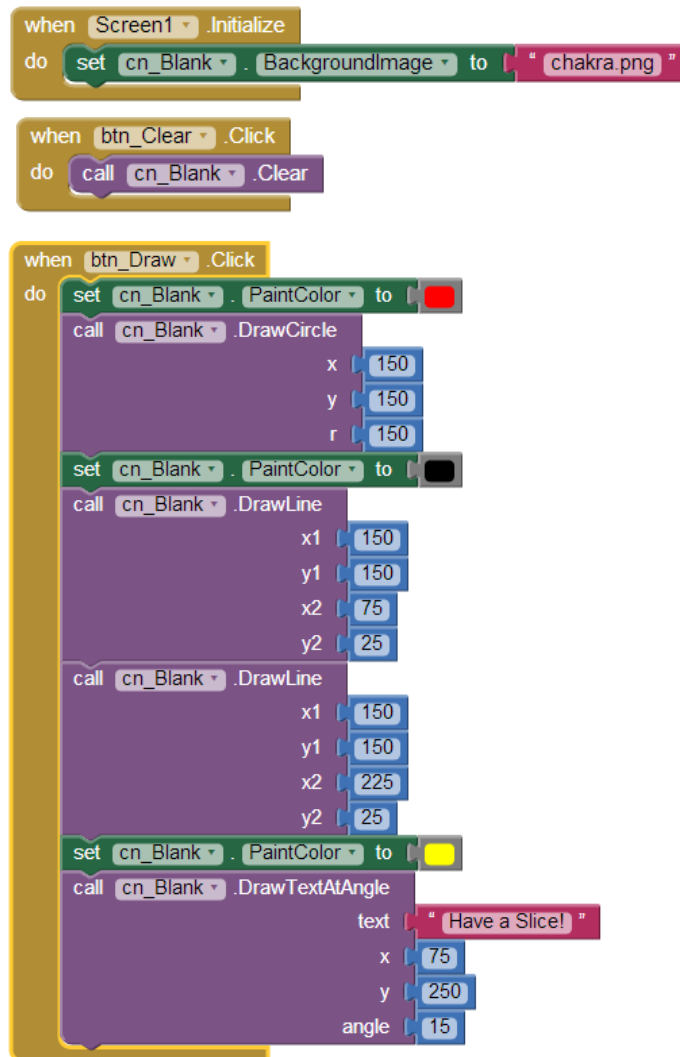


Fig 4.7 Blocks - CanvasDraw App

Here are the screen shots of CanvasDraw App as it runs on the emulator – *Initial*, after *Draw* button clicked and after *Clear* button clicked (Fig 4.8).

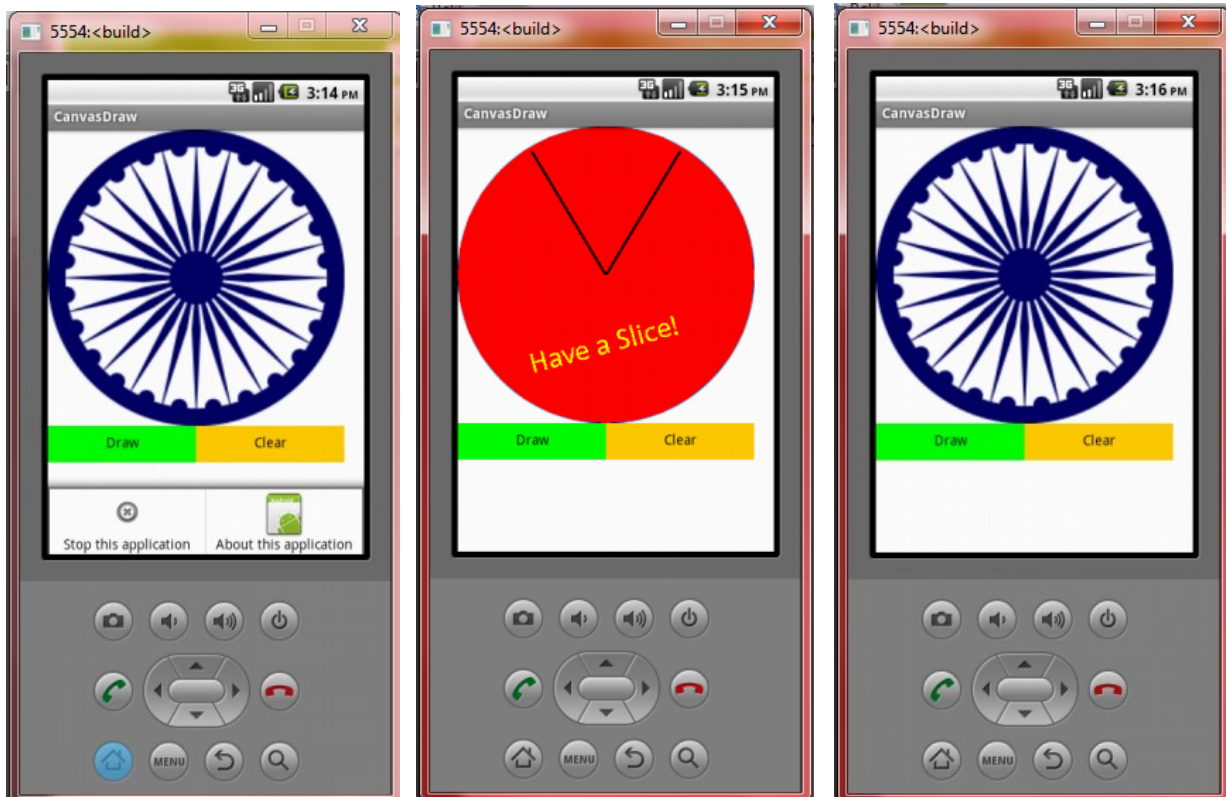


Fig 4.8 Emulator Screen shots - CanvasDraw App

## PickCourse App

Design and program an app which uses a List Picker to pick a course and a CheckBox to indicate a Lab section. Two buttons Display and Clear are used for displaying the selection and clearing. Use the ListPicker AfterPicking event to put out a message “You have picked a course”. We can also add a Texting component and a button to send a text message.



### Components

A **ListPicker** component displays a list of texts for the user to choose among and allows user to choose one. A **CheckBox** component is useful for either checking or un-checking an item. For this app, we will use the following components.

- **ListPicker** – to choose a course from a given list of courses
- **CheckBox** – to indicate a course has a Lab section
- **Buttons** – two buttons for Display and Clear

- **Horizontal Arrangement** – to hold buttons
- **Image** – to show school logo on screen



Start a **New Project** named **PickCourse**. The AI Designer will open. Set the Screen1's **Title** property to "PickCourse". You may also fill in the **AboutScreen** property to "App for Picking a Course" – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

### *Designer*

- Image **im\_FDU\_CourseFinder** (with Properties: Picture – fdu\_logo.png, Width – Fill parent, Height - 75 pixels)
- Horizontal Arrangement **HA\_PickCourse** (set Width property to Fill parent) containing
  - ListPicker **lp\_Course** (with Properties: BackgroundColor – Yellow, ElementsFromString – Biology, Chemistry, Computer Science, Creative Writing, French, History, Mathematics, FontBold, Text – Pick a Course:, Width – Fill parent)



The ListPicker has a **ElementsFromString** property where you can enter a comma separated list of items. We will set this property in the Designer. You can also set the list of items by setting the Elements property to a List in the Blocks editor.

- CheckBox **cb\_Lab** (with Properties: BackgroundColor – Cyan, FontBold, Text – Has a Lab, Width – Fill parent)
- Label **lb\_Blank1** (with Properties: FontSize - 36, No Text)
- Label **lb\_CoursePicked** (with Properties: No Text)
- Label **lb\_Blank2** (with Properties: FontSize - 36, No Text)
- Horizontal Arrangement **HA\_Display** (set Width property to Fill parent) containing
  - Button **btn\_Display** (with Properties: BackgroundColor – Magenta, FontBold, Text – Show, TextColor - White)
  - Button **btn\_Clear** (with Properties: Text – Clear:)
- Label **lb\_Blank3** (with Properties: FontSize - 36, No Text)

- Label **lb\_Display** (with Properties: FontBold, FontItalic, FontSize - 36, No Text, TextColor – Blue, Width – Fill parent)

Fig 4.9 displays the Designer - Viewer and Components.

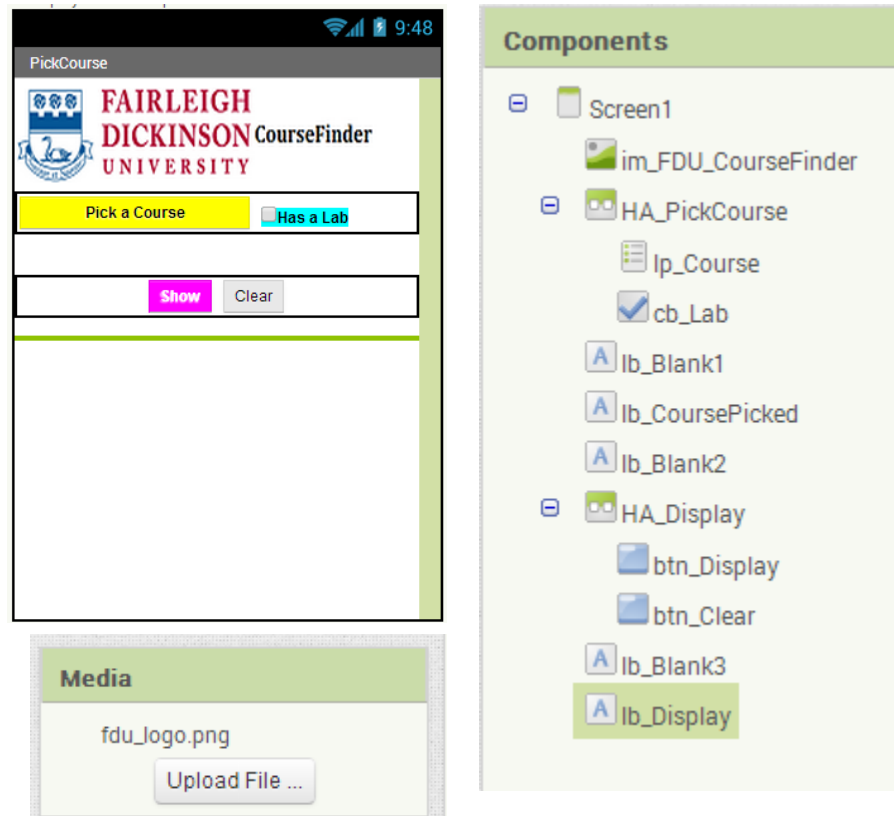


Fig 4.9 Designer - PickCourse

Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

### *Blocks*

- Locate the **lp\_Course** in the Blocks pane and drag the AfterPicking event to the workspace. Inside this add code block to do the following
  - set **lb\_CoursePicked.Text** to “You picked ” joined with **lp\_Selection.Text**. For this we use the *join* code block found in the Built-in Text block, which simply joins two strings.
- Locate the **btn\_Display** in the Blocks pane and drag the click event to the workspace. Inside this add code block to do the following
  - set **lb\_Display.Text** to 6 strings joined together: “You picked ”, **lp\_Selection.Text**, “ and ”, **cb\_Lab.Text**, “ is ”, and **cb\_Lab.Checked**. For this we use the *join* code block found in the Built-in Text block, which simply joins these 6 strings. By default the join only has 2 slots

for strings to join. You can click on the blue square in the upper left corner of join and expand it to accept more strings.



Note that **cb\_Lab.Text** is simply the Text property of the CheckBox and **cb\_Lab.Checked** is the actual check/uncheck - a Boolean value which can either be true or false.

- Locate the **btn\_Clear** in the Blocks pane and drag the click event to the workspace. Inside this call **cb\_Lab.Checked** to false. .Clear method to clear the **lb\_CoursePicked** and **lb\_Display** Text.

Fig 4.10 displays the Blocks.

In this chapter we looked at the combination of the design and program aspect of an app with 4 different apps. In the next chapter we will dig deeper into the program structure of an app.

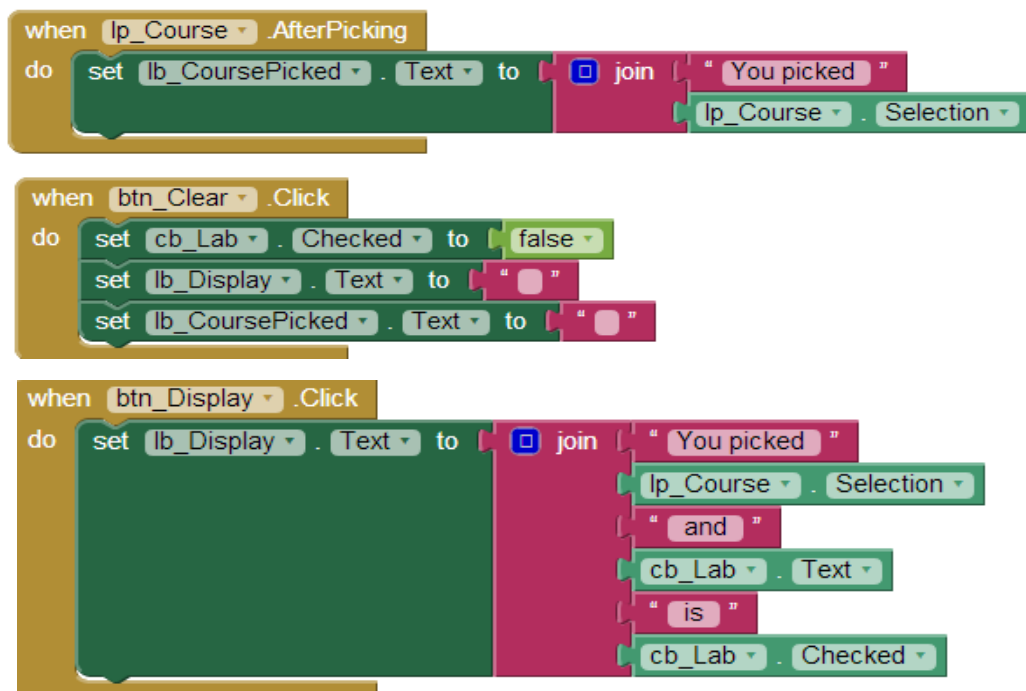


Fig 4.10 Blocks - PickCourse





## Review

- An app should accomplish what the user wants.
- Users can create something meaningful using different components and program code blocks.
- Screen has an Icon property where you can upload a icon image file, which will display on the device instead of the default App Inventor image.
- TextBox enables users to input text into your app and is visible on the screen. Also has a Hint property to let the user know what your app expects in that textbox.
- PasswordTextBox allows users to input text but this text is displayed with “\*” on the screen - useful for allowing users to input sensitive data such as passwords.
- Mutator allows blocks to expand, shrink, or even change functionality.
- Canvas provides a touch sensitive area and allows users to draw shapes and text.
- The App Inventor coordinate system is based on an (x,y) grid with upper left corner location (0,0). x gets larger to the right and y as you move down.
- ListPicker displays a list of texts for user to choose among (to choose one). List elements can be set in the Designer or the Blocks editor.
- CheckBox is useful for either checking or un-checking an item. Checked property is a Boolean value which can either be true or false.



## Chapter and Lab Summary

Please use the space below to summarize what you learnt in this chapter along with a summary of the Labs relevant to this chapter.




[illegible]

[illegible]

## Chapter 5. Program Structure

*“Any sufficiently advanced technology is indistinguishable from magic.”— Arthur C. Clarke, a British science fiction writer.*

### ICON KEY

-  Valuable information
-  Keyboard exercise
-  Review

The previous three chapters introduced you to various design components and demonstrated how to add behavior to your app simply by putting blocks together, without even writing a single line of program code.



In this chapter we will focus on the program structure and explore different ways to compose programs. From chapter 1 you know that an app is a self-contained program or piece of **software designed to fulfill a particular purpose**. When composing programs we need to keep this fact in mind.

### App Perspective

User's and programmer's view apps differently. From the user's perspective an app is something you download (for free or very cheap) from Apple App Store™, Google Play™, Windows Phone Store™, etc. to your mobile device. The app may do some work (maybe just one function) and may provide some entertainment.

From the programmer's perspective an app is like a recipe with ingredients and directions or step-by-step instructions. It is an application program or software that you use online or on mobile devices and is a series of program instructions that tell the device what to do. In this chapter we adopt this perspective when composing programs.

### Program Composition

Although programs can be complex each instruction is generally quite simple. The device starts at the beginning and works through, step-by-step, instruction by instruction, until it gets to the end.



An app is based on the **event-driven programming paradigm**, in which the flow of the program is determined by events; which could be: sensor outputs, user actions (mouse clicks, key presses), or even messages from other programs. Every component has built-in functions that handle some events. For example, the button component has a **Click** event handler. These built-in functions, called Event handlers, are programmable and you can put the instructions you want the app to execute (when the corresponding event happens) right in the event handler code block. In other words,

you can control how the app reacts to the corresponding events by programming the reactive behavior the app should take when the event occurs.

Fig 5.1 shows the architecture of an app which is composed of:

- Components - objects or elements (visible or non-visible) used to create an application.
- Variables - named containers defined in the program that store value.
- Behavior - defines how the app should respond to events which can be
  - user initiated event - for example when the user clicks a button
  - external event - for example when the phone receives a text message.
  - initialization event – when the app launches
  - timer event – when a timer fires
- Procedures - a group of one or more statements that may be called upon at any time for execution by the program.

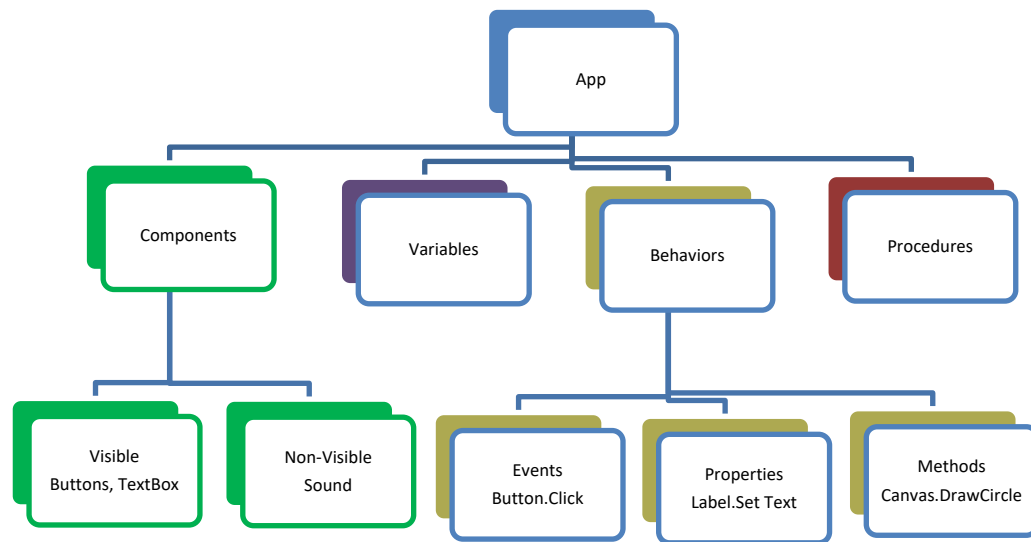


Fig 5.1 App Architecture

## Program Elements

App Inventor is based on the Object-Oriented Programming (OOP) approach which is distinct from procedural (or functional) programming approach.

The procedural programming approach is based on a top down design principle and the program itself is made up of modules or procedures and sub procedures. One of the main difficulties with this type of programming is that software maintenance can be difficult and time consuming. Whenever a change is made to the main procedure (top), it can cascade to several or all procedures (sub procedures) in the path below.

Object oriented programming is meant to address these difficulties (of procedural programming) and uses abstraction (in the form of objects) to create models based on the real world environment. Objects are capable of passing messages, receiving messages, and processing data. The aim of object-oriented programming is to increase the flexibility and maintainability of programs. Let us now see how the OOP approach works in App Inventor.

The various program elements used in an app are:

### *Object*

An object is a program's fundamental building block. It can be a visible component such as a button or a non-visible component such as a sound component. An object has properties and methods associated with it.

- **Property** – is a characteristic of an object – for example **btn\_Start.Text** is the text property of the **btn\_Start** object.
- **Method** – is an action that an object can perform – for example **btn\_Start.click** is the click method of the **btn\_Start** object.

### *Statement*

A statement is an instruction that performs a program task. For example – the set **lb\_Display.Text** to Hello which sets the Text property of the label object to the text Hello.

### *Procedure*

A procedure is a group of one or more statements that may be called upon at any time for execution by the program. We will see procedures in a later chapter. Groups of statements can be written as a procedure which can be called many times from different places.

### *Variable*

A variable is a named container defined in the program that stores a value.

### *Operator*

An operator is an arithmetical symbol. For example, + addition, - subtraction, \* multiplication, and / division operators

### *Comment*

A comment is optional and describes the purpose of a statement. It is a good idea to incorporate comments in your program since it makes your code easy to understand by others.

## Control Flow

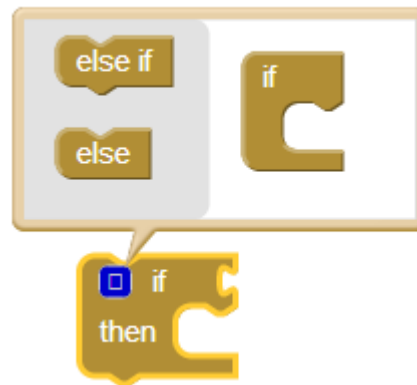
The statements inside the code blocks are generally executed step-by-step from top to bottom. However, you may want to perform different actions in different situations based on user input. You thus need the ability to control the flow of your program letting it make decisions on what code to execute next driven by a decision which may be based on user input for example.

The Control drawer in the **Built-in** tab in the Blocks pane of the Blocks Editor houses a number of blocks which enable your app to perform different actions depending upon the results of a conditional test. The given statements are only executed when the tested condition is true, otherwise these statements are skipped. This is easily achieved using the *if-then* block from the **Built-in** Control tab. The conditional test can be a simple test as in the **ValidateNumber** app below, or, it can be a complex one using

several layers of logic with  and  blocks from the Logic drawer.



The *if-then* block from the **Built-in** Control tab has options for providing alternatives



using *else* and *else if* making it possible to nest multiple conditions as demonstrated in the **AssignGrade** app.

## ValidateNumber App

Design and program an app which requests user to enter a positive number and then checks if the entered number is valid or not. This app uses a *if-then* test block from the **Built-in** Control tab in the Blocks pane of the Blocks Editor which we saw in the Chapter 3 earlier. This app demonstrates the use of a variable for setting the appropriate message to display on validation and also shows how to add comments to the various blocks.



The **if** statement allows you to control if a program enters a section of code or not based on whether a given condition is true or false. One of the important functions of the **if** statement is that it allows the program to select an action based upon the user's input. For example, we will use an **if** statement to check whether the user has entered a

valid number. For this app, we will use the following components.

### Components

- **Textbox** – to let user enter a number
- **Button** – to show validation message
- **Horizontal Arrangement** – to hold button and textbox
- **Label** – to display validation message on screen



Start a **New Project** named **ValidateNumber**. The AI Designer will open. Set the Screen1's **Title** property to “ValidateNumber”. You may also fill in the **AboutScreen** property to “App for Validating a Number” – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

### Designer

- Horizontal Arrangement **HA\_EnterNumber** (set Width property to Fill parent) containing
  - TextBox **tb\_EnterNumber** (with Properties: BackgroundColor – Yellow, Hint – Enter a number)
  - Button **btn\_Validate** (with Properties: BackgroundColor – Magenta, FontBold, Text – Validate, TextColor – White, Width – Fill parent)
- Label **lb\_Display** (with Properties: FontBold, FontItalic, FontSize - 18, No Text, TextColor – Blue, Width – Fill parent)

Fig 5.2 displays the Designer - Viewer and Components.

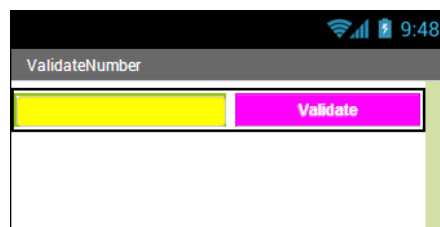
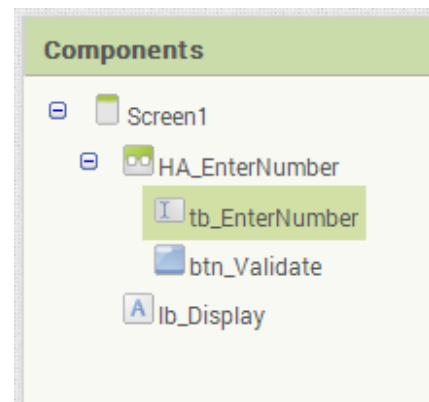



Fig 5.2 Designer - ValidateNumber



Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

### Blocks

- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag the **initialize global name to** block to the workspace. Change name to **msg** and set it to “Number is Invalid!”. Later we will change **msg** to appropriate message.
- Right click on this block and select Add Comment. Click on  and add a comment in the box as shown in Fig 5.3.

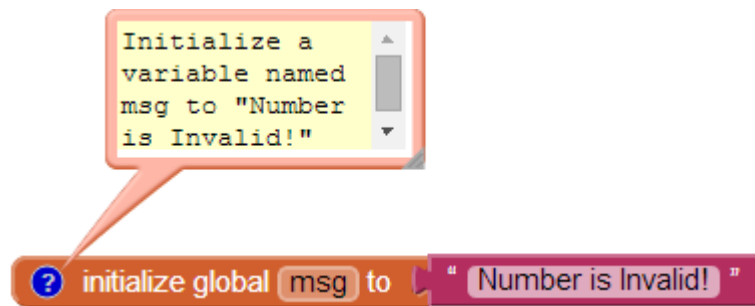
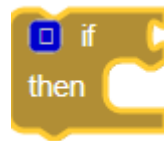
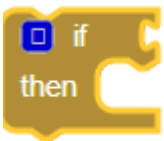



Fig 5.3 Add Comment

- Locate the **btn\_Validate** in the Blocks pane and drag the click event to the workspace. Inside this add code block to do the following:
  - Locate the **Control** drawer in the Blocks pane under the Built-in tab



and drag the  to the workspace. This is where we do the check if number is valid or not.

- Drag  from the **Math** drawer in the Blocks pane to fit into the if and change = to > to perform the greater than operation on **tb\_EnterNumber.Text** and number **0**.
- In the then part – set the variable **msg** to “Number is Valid!”



- After **msg** has been set appropriately, set **lb\_Display.Text** to **msg** and reset **msg** back to “Number is Invalid!” for next click.

Fig 5.4 displays the Blocks with various comments.

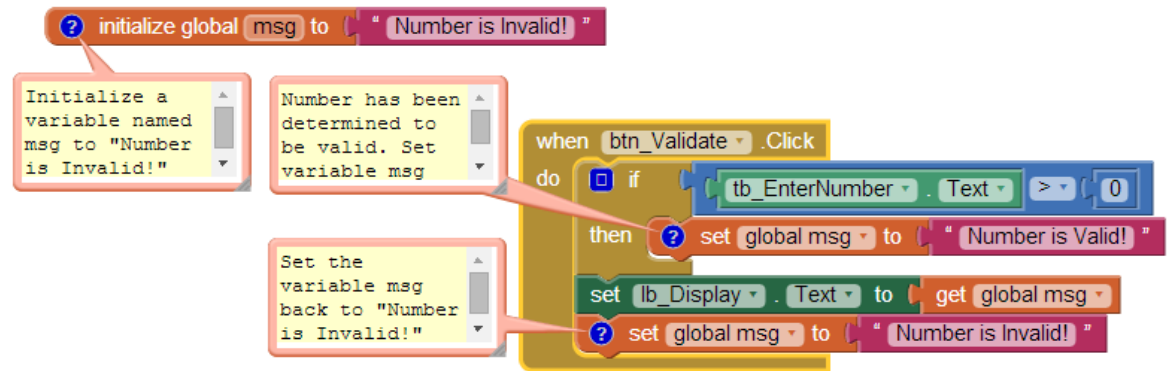


Fig 5.4 Blocks - ValidateNumber

## AssignGrade App

Design and program an app which requests user to enter a score and then displays a letter grade based on that score using the criteria shown. If the score is greater than or equal to 90 then grade is “A”. Otherwise if the score is between 80 and 90 then grade is “B” and so on. For a score below 60 assign grade “F”.

Score	Grade
$\geq 90$	A
$\geq 80$ and $< 90$	B
$\geq 70$ and $< 80$	C
$\geq 60$ and $< 70$	D
$< 60$	F



This app demonstrates the use of **nested if** as well as logic blocks to test multiple conditions for setting the appropriate letter grade. These nested **if** statements allow you to control if a program enters a section of code (or not) based on the multiple conditions. One of the important functions of the nested **if** statement is that it allows the program to select an action based upon the user's input. For example, we will use nested **if** statement to check whether the score is within a certain range. For this app, we will use the following components.

### Components

- **Textbox** – to let user enter a score
- **Button** – to assign a letter grade
- **Horizontal Arrangement** – to hold button and textbox
- **Label** – to display the letter grade on screen



Start a **New Project** named **AssignGrade**. The AI Designer will open. Set the Screen1's **Title** property to "AssignGrade". You may also fill in the **AboutScreen** property to "App for Assigning a Letter Grade based on a score." – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

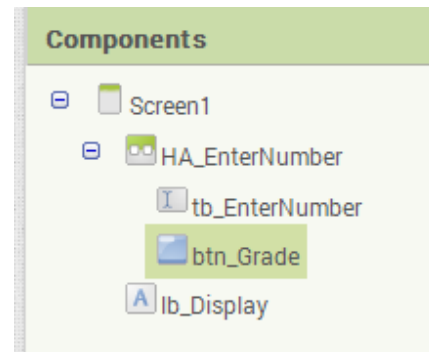
### *Designer*

- Horizontal Arrangement **HA\_EnterNumber** (set Width property to Fill parent) containing
  - TextBox **tb\_EnterNumber** (with Properties: BackgroundColor – Yellow, Hint – Enter score)
  - Button **btn\_Grade** (with Properties: BackgroundColor – Magenta, FontBold, Text – Grade, TextColor – White, Width – Fill parent)
- Label **lb\_Display** (with Properties: FontBold, FontItalic, FontSize - 18, No Text, TextColor – Blue, Width – Fill parent)

Fig 5.5 displays the Designer - Viewer and Components.



Fig 5.5 Designer - AssignGrade

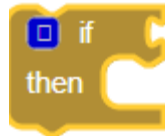


Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

### *Blocks*

- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag the **initialize global name to** block to the workspace. Change name to **score** and set it to a number 0.
- Locate the **btn\_Grade** in the Blocks pane and drag the click event to the workspace. Inside this add code block to do the following:

- Set the global variable **score** to the value which the user entered in the textbox **tb\_EnterNumber.Text**
- Set **lb\_Display.Text** to “Your Grade is “
- Locate the **Control** drawer in the Blocks pane under the Built-in tab



and drag to the workspace. Fig 5.6 shows the check for assigning the letter grade as per the requirements. Fig 5.7 shows a flow chart to help you understand the logic behind these blocks.



Fig 5.6 Blocks\_AssignGrade

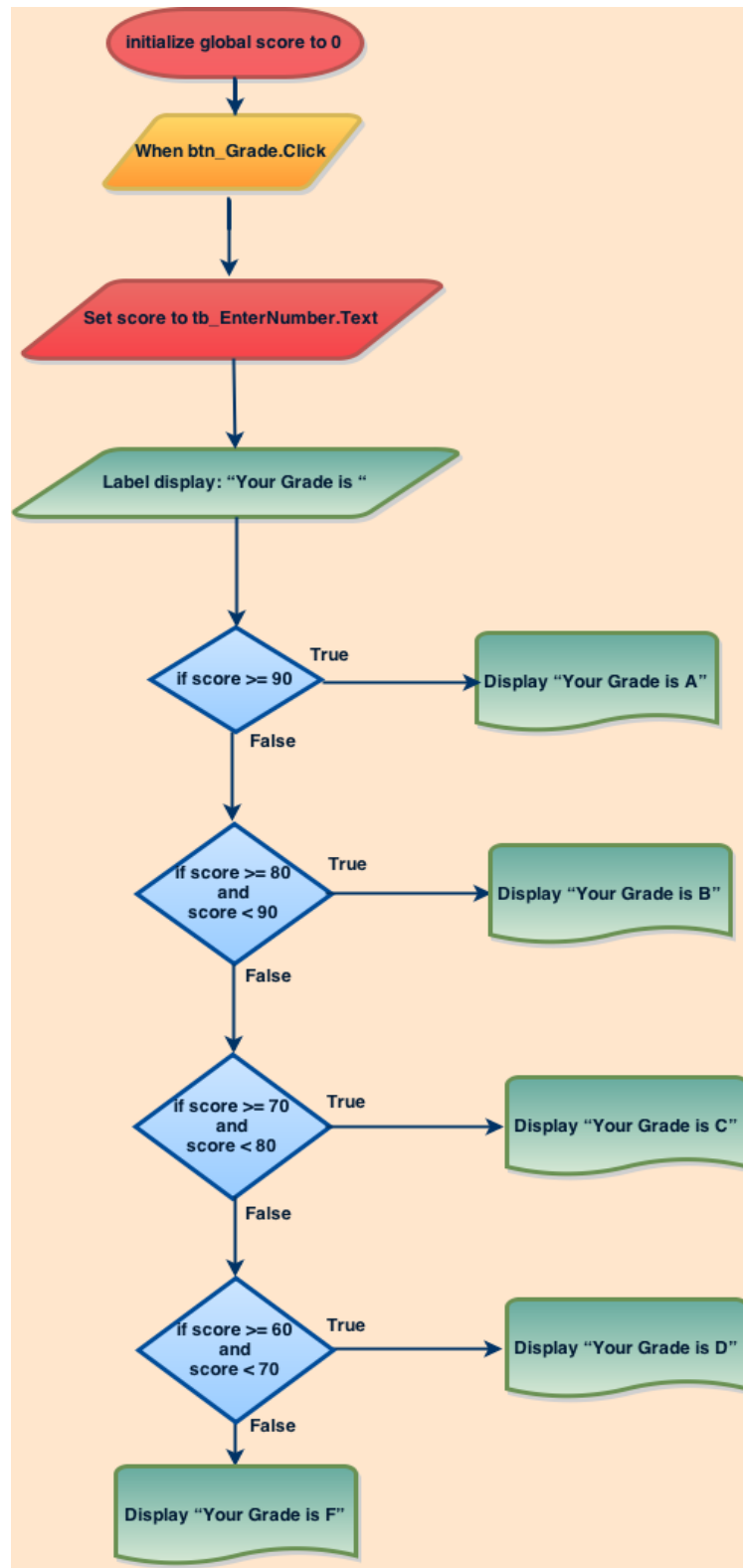


Fig 5.7 FlowChart - AssignGrade

## Repetition With Loops

The statements inside the code blocks are generally executed step-by-step from top to bottom. However, you may want to perform some actions repeatedly based on user input or another criterion. You thus need the ability to perform repetitions in a loop without repeating the code in your blocks. In order to avoid repeating endlessly you will need to determine when to terminate the loop and stop the repetition.

The Control drawer in the **Built-in** tab in the Blocks pane of the Blocks Editor houses a number of blocks which enable your app to perform repeatedly by programming loops. Loop termination depends upon the results of a conditional test. The given statements are only executed when the tested condition is true, otherwise the loop terminates and these statements are skipped. This is easily achieved using the *for each* block from the **Built-in** Control tab. The conditional test can be a simple test as in the **SumOfNumbers** app below.

## SumOfNumbers App

Design and program an app which requests user to enter a positive number  $n$  and computes the sum of  $1+2+3+\dots+n$  where  $n$  is the number the user enters. This app uses a *for each* block from the **Built-in** Control tab in the Blocks pane. The program loops over the statements  $n$  times computing the cumulative sum.



The *for each* statement allows you to control how repetition is performed and executes the statements in the do section for each numeric value in the range from start to end, increasing the value for  $i$  by step each time. Usually the given variable name  $i$  is used to refer to the current value. The loop is repeated  $n$  times with  $i$  starting with 1 and ending with  $n$ . For this app, we will use the following components.

### Components

- **Textbox** – to let user enter a number
- **Button** – to show the cumulative sum
- **Horizontal Arrangement** – to hold button and textbox
- **Label** – to display sum of numbers from 1 to  $n$  on screen



Start a **New Project** named **SumOfNumbers**. The AI Designer will open. Set the Screen1's **Title** property to "SumOfNumbers". You may also fill in the **AboutScreen** property to "App for Summing numbers from 1 to  $N$ " – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

## *Designer*

- Horizontal Arrangement **HA\_EnterNumber** (set Width property to Fill parent) containing
  - TextBox **tb\_EnterNumber** (with Properties: BackgroundColor – Yellow, Hint – Enter a number)
  - Button **btn\_Sum** (with Properties: BackgroundColor – Magenta, FontBold, Text – Sum, TextColor – White, Width – Fill parent)
- Label **lb\_Display** (with Properties: FontBold, FontItalic, FontSize - 18, No Text, TextColor – Blue, Width – Fill parent)

Fig 5.8 displays the Designer - Viewer and Components.

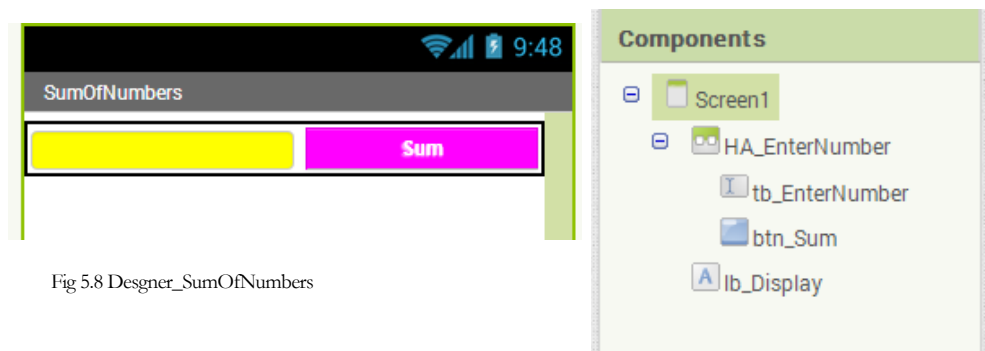


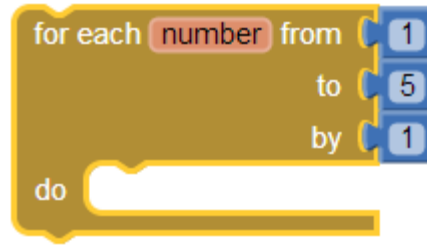
Fig 5.8 Designer\_SumOfNumbers

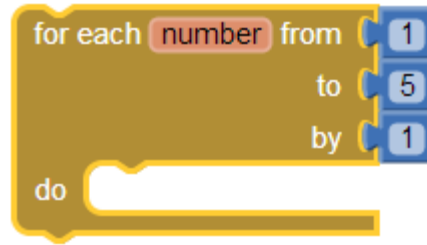
Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

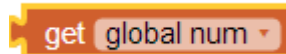
## *Blocks*

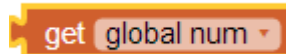
- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag two **initialize global name to** to the workspace. Change name to **num** and set it to a number 0 for one and change the name to **sum** and set it to 0.
- Locate the **btn\_Sum** in the Blocks pane and drag the click event to the workspace. Inside this add code block to do the following:
  - Set the global variable **num** to the value which the user entered in the textbox **tb\_EnterNumber.Text**
  - Set **lb\_Display.Text** to 3 strings joined together “Sum of numbers from 1 to ”, **get global num**, “ is ”

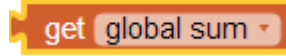
- Locate the **Control** drawer in the Blocks pane under the Built-in tab



and drag  to the workspace. Change **number** to **i**. Leave the **from** value to 1 and set the **to** value to



. Leave the **by** value to 1. In the **do** part write the code to perform the cumulative sum by adding **i** to the



as shown in Fig 5.9.

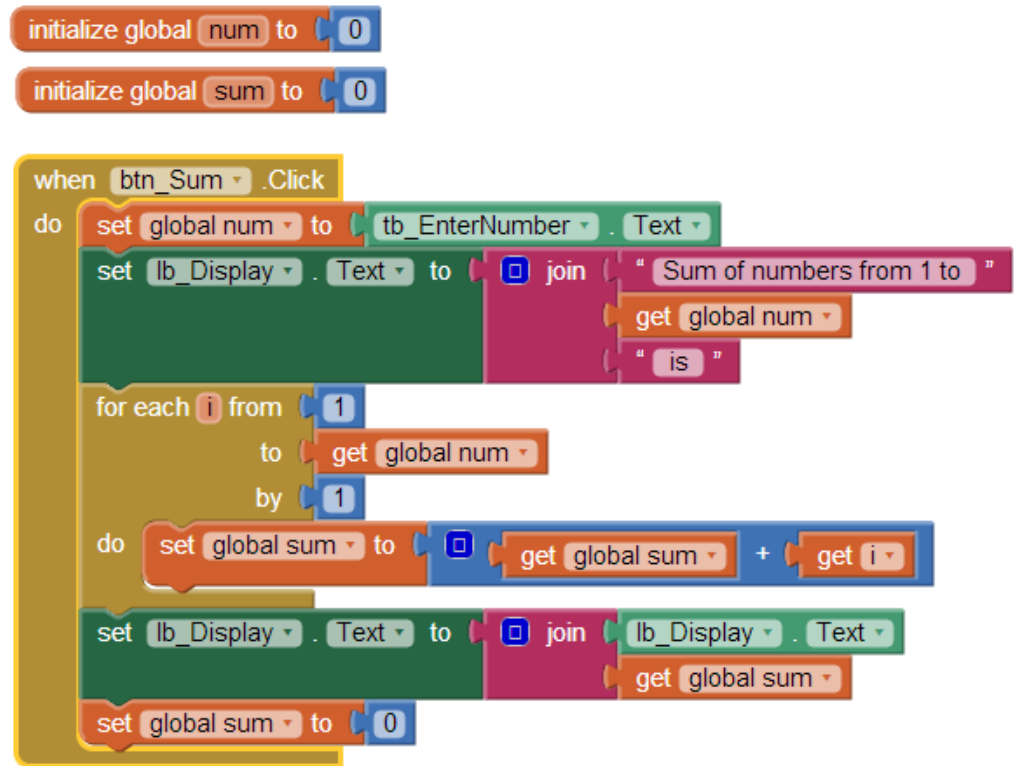


Fig 5.9 Blocks - SumOfNumbers

Fig 5.10 shows a flow chart to help you understand the logic behind the **for each** block.

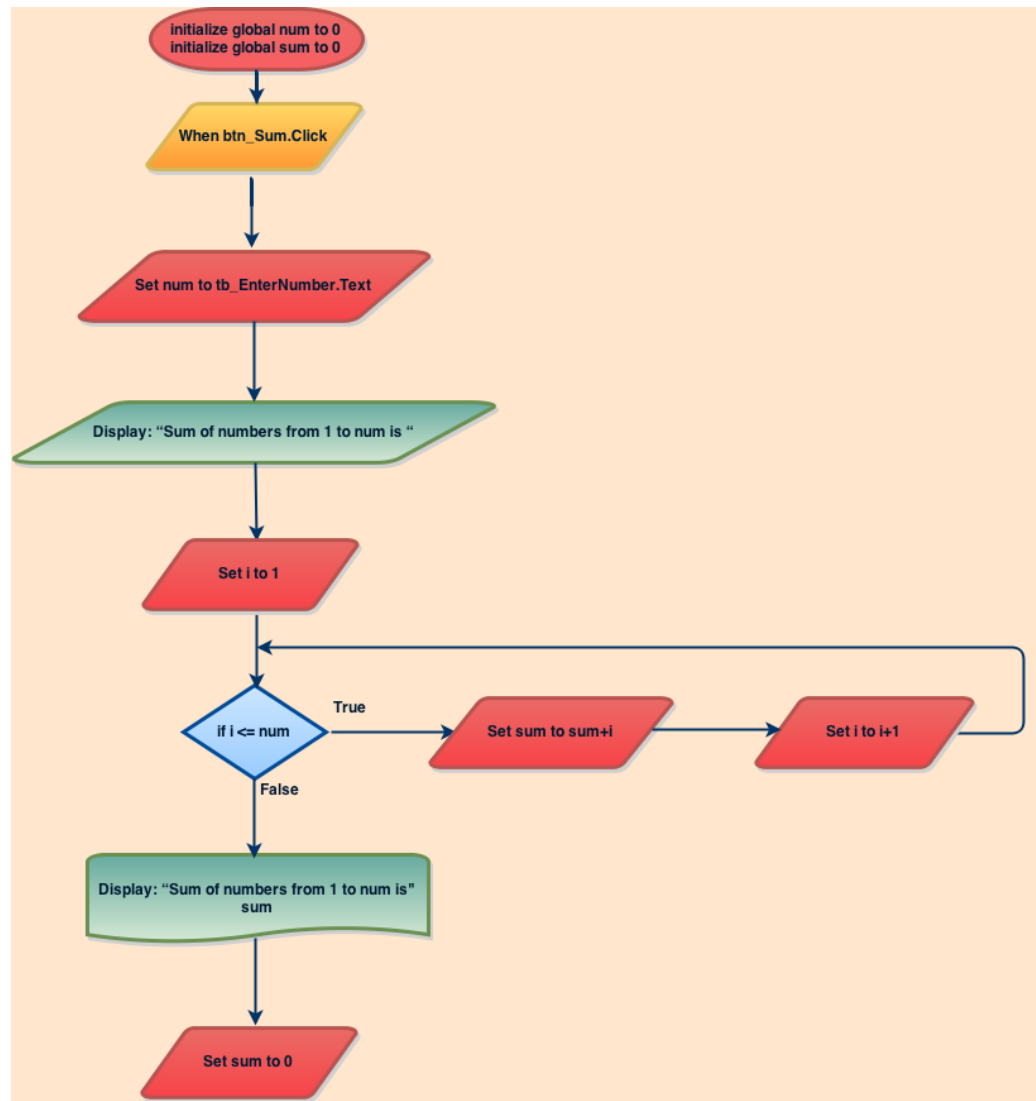


Fig 5.10 FlowChart - SumOfNumbers

The next section demonstrates the use of a **while** loop instead of the **for each** to perform the cumulative summation.

## SumOfNumbers\_While App

We re-design the SumOfNumbers app to use a **while** loop (instead of the **for each**) block from the **Built-in** Control tab in the Blocks pane. Just like the previous app, this loops over the statements n times computing the cumulative sum.



The **while** statement allows you to control how repetition is performed and executes the statements in the do section while a test condition remains true. The while loop ends when the test is false and the action given in the do section is no longer performed. This while needs to be primed outside the loop by setting a variable which



is used to decide if loop should continue or end. For this app, we will use the same components as the previous app.

### *Components*

- **Textbox** – to let user enter a number
- **Button** – to show the cumulative sum
- **Horizontal Arrangement** – to hold button and textbox
- **Label** – to display sum of numbers from 1 to n on screen



Start a **New Project** named **SumOfNumbers\_While**. The AI Designer will open. Set the Screen1's **Title** property to “SumOfNumbers\_While”. You may also fill in the **AboutScreen** property to “App for Summing numbers from 1 to N using the While Loop” – this will be displayed when user clicks on **About this Application** when your app runs. (Instead of starting a new project, a faster way would be to save the previous project as SumOfNumbers\_While and make some changes to the blocks).

Using the AI Designer add the following components to Screen1 and change some of the properties.

### *Designer*

- Horizontal Arrangement **HA\_EnterNumber** (set Width property to Fill parent) containing
  - TextBox **tb\_EnterNumber** (with Properties: BackgroundColor – Yellow, Hint – Enter a number)
  - Button **btn\_Sum** (with Properties: BackgroundColor – Magenta, FontBold, Text – Sum, TextColor – White, Width – Fill parent)
- Label **lb\_Display** (with Properties: FontBold, FontItalic, FontSize - 18, No Text, TextColor – Blue, Width – Fill parent)

Fig 5.11 displays the Designer - Viewer and Components.

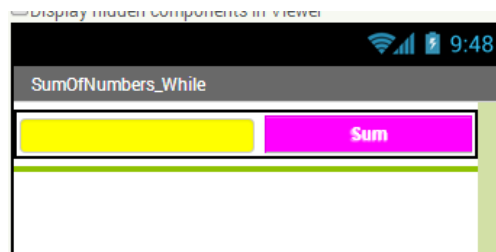
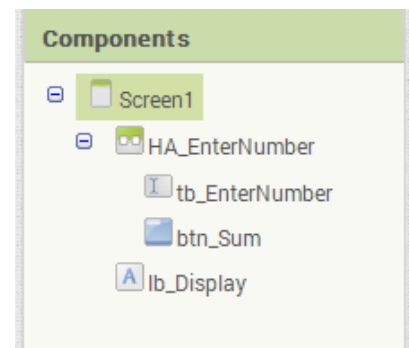

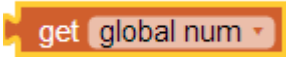
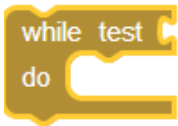
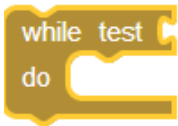


Fig 5.11 Designer - SumOfNumbers\_While



Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

## Blocks

- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag three  to the workspace. Change name to **num** and set it to a number 0 for one and change the name to **sum** and set it to 0 just like in the previous app. For the third initialize change the name to **i** and set it to 1. This variable **i** will be used to test the while loop execution and is the “priming” value for the while loop. Also note that this variable will need to be updated before looping back.
- Locate the **btn\_Sum** in the Blocks pane and drag the click event to the workspace. Inside this add code block to do the following:
  - Set the global variable **num** to the value which the user entered in the textbox **tb\_EnterNumber.Text**
  - Set **lb\_Display.Text** to 3 strings joined together “Sum of numbers from 1 to “, , “ is “
  - Locate the **Control** drawer in the Blocks pane under the Built-in tab  and drag  to the workspace. In the **test** section add code to check **i <=** the number entered by the user. The **do** section has the code to do the cumulative sum and the increment of the variable **i** as shown in Fig 5.12.

Think about what would happen if **i** is not incremented – the test would always be true and loop will not end – which would result in an infinite loop.

In this chapter we looked at the program structure, composition and elements of apps and some ways to alter the flow of the program. We also saw how to repeatedly execute some program statements using **for each** and **while** statements. In the next chapter we will look into **Lists** and the use of **for each item in list** code block.

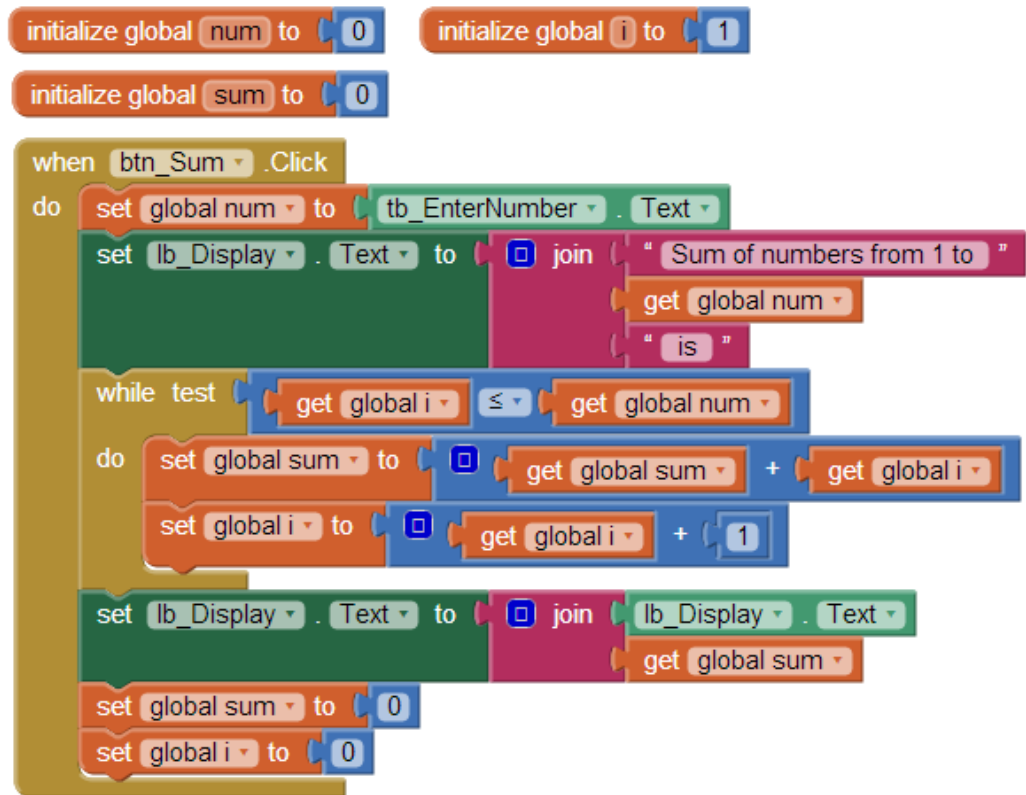


Fig 5.12 Blocks - SumOfNumbers\_While



## Review

- App is a self-contained program designed to fulfill a particular purpose.
- Programs can be complex but each instruction is generally quite simple; the device starts at the beginning and works through, step-by-step, instruction by instruction, until it gets to the end.
- User's and programmer's view apps differently.
- App is based on the event-driven programming paradigm, with events such as sensor outputs, user actions (mouse clicks, key presses) etc.
- Event handlers are built-in functions which are programmable. The instructions you want the app to execute (when the corresponding event happens) are right in the event handler code block.
- The architecture of an app is composed of components, variables, behaviors and procedures. App Inventor is based on the Object-Oriented Programming (OOP), which uses abstraction (in the form of objects) to create models based on the real world environment. OOP increases the flexibility and maintainability of programs.
- AI gives you the ability to control the flow of your program, to make decisions on what code to execute next based on user input for example. The if statement allows you to control if a program enters a section of code or not based on whether a given condition is true or false. You can use nested if to test multiple conditions.

- The for each and while statements give you the ability to perform repetitions in a loop without repeating the code in your blocks. In order to avoid endless repetition, you need to carefully determine when to terminate the loop and stop the repetition.



## Chapter and Lab Summary

Please use the space below to summarize what you learnt in this chapter along with a summary of the Labs relevant to this chapter.

[illegible]




[illegible]

## Chapter 6. Lists

*“The list is the origin of culture. It's part of the history of art and literature. What does culture want? To make infinity comprehensible. It also wants to create order -- not always, but often.”– Umberto Eco, an Italian writer and philosopher.*

---

### ICON KEY

- 
-  Valuable information
  -  Keyboard exercise
  -  Review
- 

The previous chapter introduced you to the program structure, composition and elements of apps and demonstrated ways to alter the flow of your program. In this chapter we will focus on an important data structure – **list**, which is very useful for storing a series of items or multiple values as opposed to a single value in a variable. For example you may keep a list of high scores in a game or a list of your friends in your Facebook app, etc.

Most programming languages use lists to create and manipulate different sets of values and/or elements. A list is also known as an **array** in some programming languages. A **list** can store data such as **numbers** (telephone numbers of your contacts in your phone), **text** (answers for a Trivia app), or even **colors** (palette of colors for a Paint app).



The **Lists** drawer (found in the **Built-in** tab in the Blocks pane of the Blocks Editor) houses a number of list blocks which enable your app to create and manipulate lists. There are two ways to create an empty list – use **create empty list** or use **make a list** without any elements.

Other blocks such as **is a list?**, **is in list?**, and **is list empty?** provide functionality to query lists and return boolean values (true or false) as result. Functions which return integer values include **length of list** and **position in list**. List blocks also include functions to add item to a list, remove item from a list, and select a particular or a random item from a list. The function **add items to list** adds an item to the end of the list, **insert list item** inserts an item into the list at a specified position. You can also add items from a list to the end of another list using the **append to list function**. The Lists drawer also includes functions to separate lists into a string of comma-separated values (CSV) or create lists from CSV.

### MoleMashList App

We now create a MoleMashList app which is similar to the Whac-a-Mole™ arcade game where a mole randomly pops out of one of five fixed holes and the goal of the game is to force the mole back into its hole by tapping it. The score is increased by one point each time the mole is successfully tapped.

---



We will first create an empty list and then populate it with elements which represent the 5 holes on the screen. The **Clock** component is used to control the mole's movement. We will use the **Sprite Z-layering** to ensure that the mole appears in front of the hole. For this app, we will use the following components.

### Components

- **ImageSprites**– to represent a mole and 5 holes
- **Canvas** – a touch-sensitive surface to house the mole and holes
- **Clock** – to control the mole's movement
- **Labels**– to display the game score
- **Horizontal Arrangement** – to hold the score label



Start a **New Project** named **MoleMash\_List**. The AI Designer will open. Set the Screen1's **Title** property to "MoleMash\_List." You may also fill in the **AboutScreen** property to "App for Whack-A-Mole arcade game using Lists" – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

### Designer

- Canvas **cnv\_moleGame** (set Width and Height property to 320 pixels) containing 6 ImageSprites (5 holes in two rows and one mole below)
  - **is\_hole1** (with Properties: X – 20, Y – 60, Z – 1)
  - **is\_hole2** (with Properties: X – 130, Y – 60, Z – 1)
  - **is\_hole3** (with Properties: X – 240, Y – 60, Z – 1)
  - **is\_hole4** (with Properties: X – 75, Y – 140, Z – 1)
  - **is\_hole5** (with Properties: X – 185, Y – 140, Z – 1)
  - **is\_mole** (with Properties: X – 140, Y – 220, Z – 2). Note: the mole image sprite has its **Z value set to 2** (higher than the Z value of the hole image sprites).
- Horizontal Arrangement **HA\_EnterNumber** containing 2 labels
  - Label **lb\_ScoreTextLabel** (with Properties: Text – Score: )
  - Label **lb\_Score** (with Properties: Text – 0)
- Sound **snd\_Buzzer** from the Media tab of the Palette pane. Note that this is a non-visible component.
- Clock **clk\_MoleClock** from the Sensors tab of the Palette pane (with Properties: TimerAlwaysFires – checked, TimerEnbaled– checked and

TimeInterval – 1000). This non-visible component fires every 1000 milliseconds or every second.

- Upload two image files (hole.png and mole.png) in the Media pane.

Fig 6.1 displays the Designer - Viewer and Components

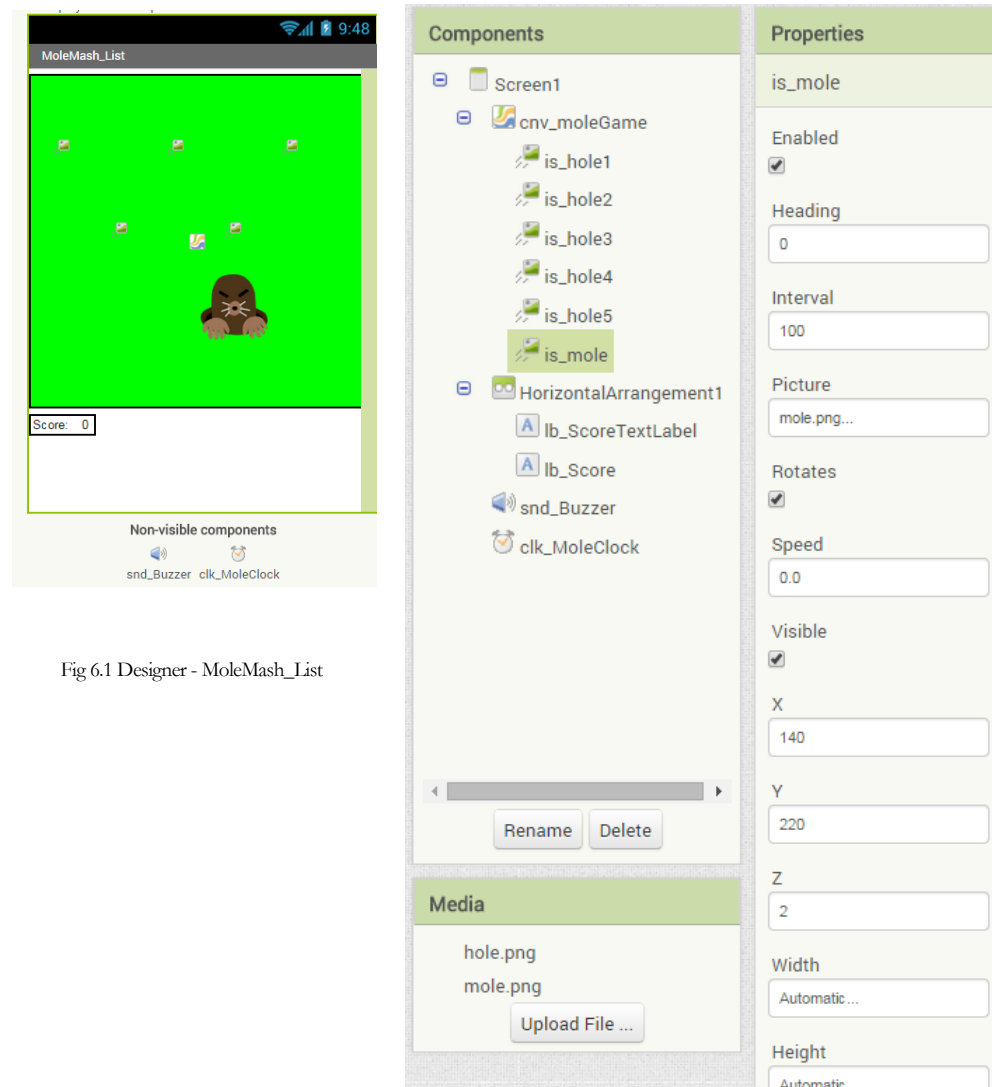


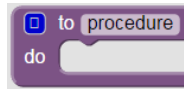
Fig 6.1 Designer - MoleMash\_List

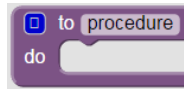
Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

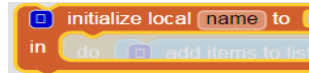
### *Blocks*

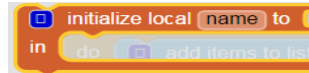
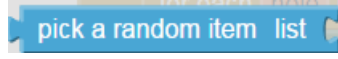
- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag three **initialize global name to** to the workspace. Change name to **holes** and set it to **create empty list** from the **Lists** drawer. Now we have a empty list named **holes**. We will populate this list shortly.



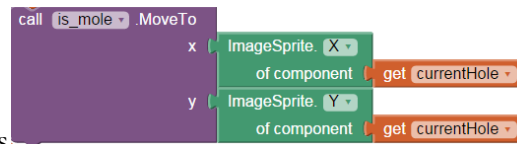


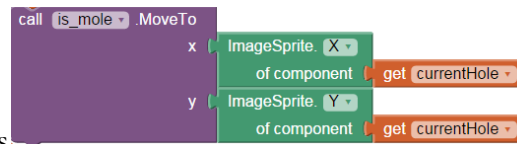
- Drag the  from the **Procedures** drawer in the Blocks pane under the Built-in tab to the workspace and change procedure to **moveMole**. Inside this add code block to do the following:

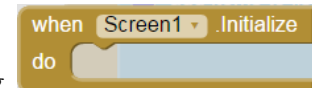


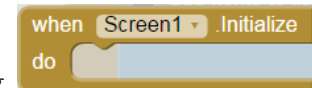
- Drag  from the **Variables** drawer and set a local variable **currentHole** to a random item from the global **holes** list using  from the **Lists** drawer.

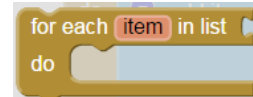
- Locate the imagesprite **is\_mole** in the Blocks pane under Screen1-**cnv\_moleGame** and call MoveTo function and set the X and Y

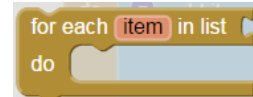


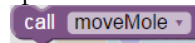
components . This will move the mole to the current hole (a randomly picked hole from the holes list).

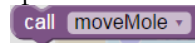


- Locate Screen1 in the Blocks pane and drag  to the workspace. This is where we will populate the **holes** list by using add items to list as shown in the Blocks in Fig 6.2. We need to use the appropriate hole components from the imagesprite **is\_hole** in the Blocks

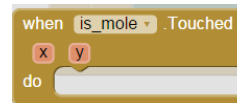


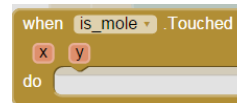
pane under Screen1 - **cnv\_moleGame**. Next use  to set the picture property of the 5 hole imagesprites to hole.png (previously uploaded in the Media pane of the AI designer). Finally add




 from the **Procedures** drawer (in the Blocks pane under the Built-in tab) to move the mole to another hole.

- Locate the imagesprite **is\_mole** in the Blocks pane under Screen1-



**cnv\_moleGame** and call  to do the following when the mole is touched (as shown in Fig 6.2)

- update the score,
- make the buzzer vibrate, and
- move the mole to another hole by .

- Locate the Clock component **clk\_MoleClock** in the Blocks pane under Screen1 and drag  event to the workspace and add

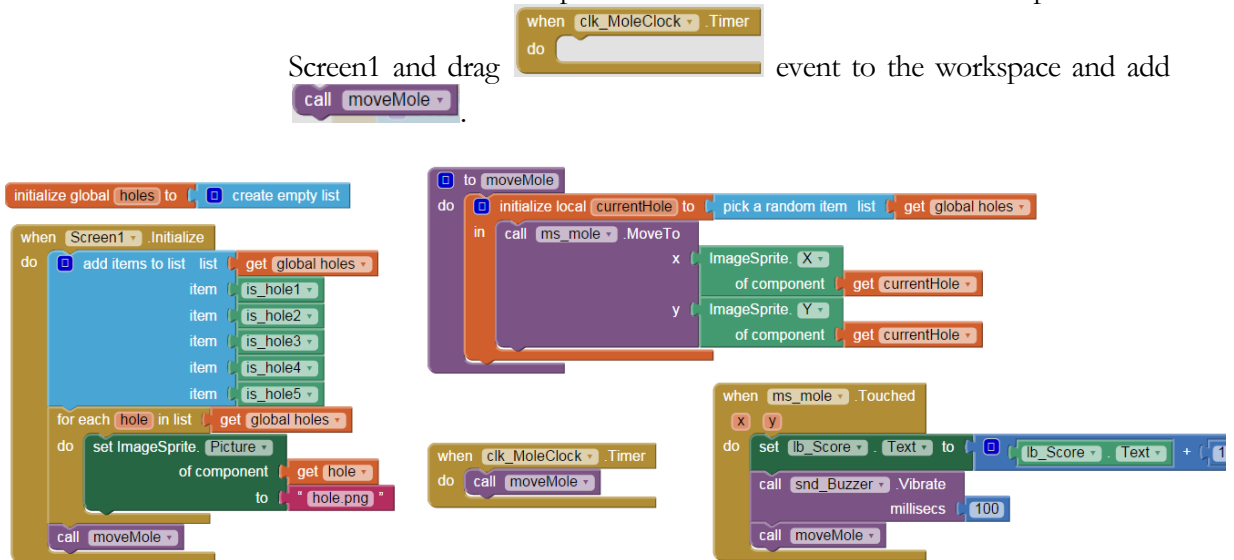


Fig 6.2 Blocks – MoleMash\_List

## Trivia App

We now create a Trivia app for a Question/Answer game for a college campus.



### Components

We will first create three lists – questions list, answers list and pictures list. Then we will build a Trivia game with Questions/Answers about various buildings/structures on a college campus. For this app, we will use the following components.

- Upload six image files (we will have questions about these six structures) in the Media pane
- **Image** – to display a picture related to the question
- **Horizontal Arrangements** – to hold the textual answer to the question the user enters and the Submit and Next buttons for submitting the answer or for the next question.
- **Labels**– to display the question, answer and correct/incorrect



Start a **New Project** named **FDU\_BuildingsTriviaGame**. The AI Designer will open. Set the Screen1's **Title** property to “FDU Buildings Trivia Game.” You may also fill in the **AboutScreen** property to “App for a Campus Trivia game using Lists” – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

## Designer

- Image **img\_FDU** (with Properties: Width – Fill parent, Height – 285 pixels, Picture: Mansion.png (initially we use one of the building pictures which we uploaded in the Media pane, this will change later on in the code).
- Label **lb\_QuestionLabel** (with Properties: Text – Question: )
- Horizontal Arrangement **HA\_Answer** containing
  - Label **lb\_AnswerLabel** (with Properties: Text – Answer: )
  - TextBox **tb\_Answer** (with Properties: Hint – Enter a Building Name, with no text)
- Label **lb\_RightWrong** (with Properties: Text – Correct/Incorrect)
- Horizontal Arrangement **HA\_SubmitNext** containing 2 buttons
  - Button **btn\_Submit** (with Properties: Text – Submit)
  - Button **btn\_Next** (with Properties: Text – Next)

Fig 6.3 displays the Designer - Viewer and Components

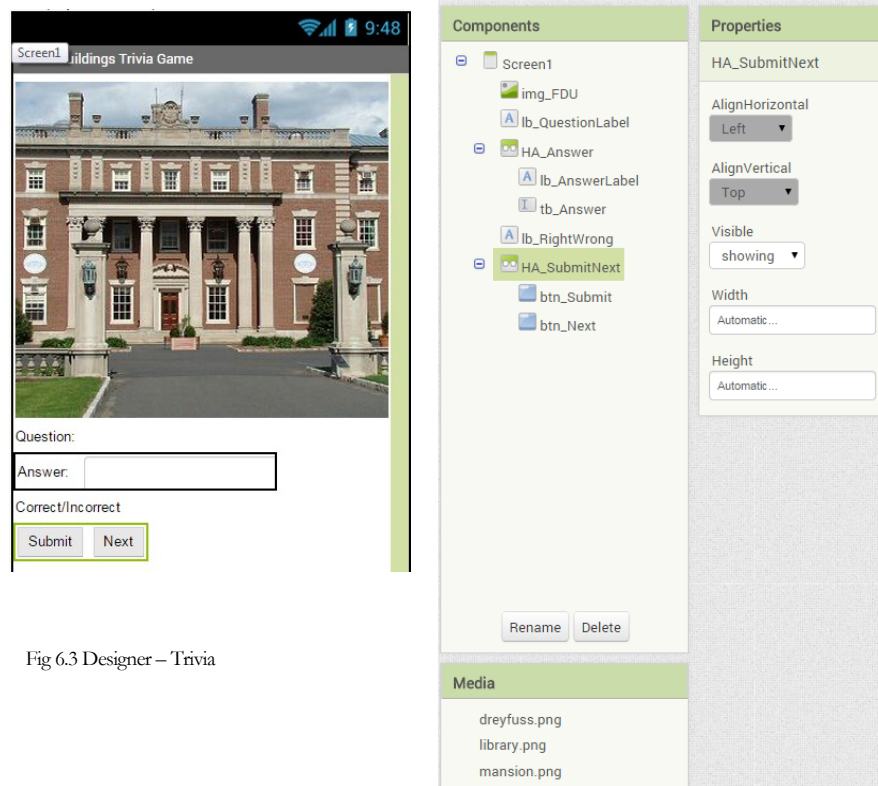

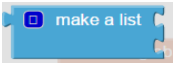
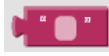


Fig 6.3 Designer – Trivia

Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

## Blocks

- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag three  to the workspace. Change name to **questionList** and set it to  from the **Lists** drawer. Now populate this list with six different questions using  and the mutator to expand. Repeat the same with the **answerList** and the **pictureList** as shown below in Fig 6.4.

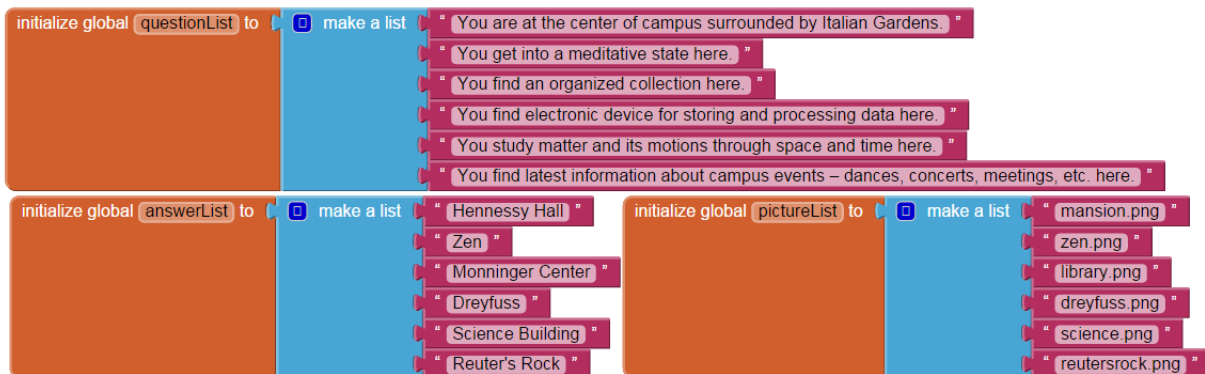
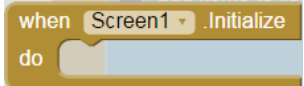



Fig 6.4 InitializeLists – Trivia

- Locate Screen1 in the Blocks pane and drag  to the workspace. This is where we will pop the first question by setting the text property of the **lb\_QuestionLabel** to the selected first question in the **questionList** as shown in Fig 6.5 below.
- Initialize a global **currentQindex** to 1 for the first question.
- Locate the **btn\_Submit** in the Blocks pane and drag the click event to the workspace. Inside this add code block to do the following:
  - use the **if-then** block with else added with the mutator from the **Control** drawer in the Blocks pane under the Built-in tab to check if the answer entered in the **tb\_Answer** matches the answer of the current question. If it matches then set the properties of the **lb\_RightWrong** to: Text – Correct, Background Color – Blue, and Text Color – White; otherwise set properties to: Text – Incorrect, Background Color – Red, and Text Color – White.

- Locate the **btn\_Next** in the Blocks pane and drag the click event to the workspace. Inside this add code block to do the following:
  - Clear out the answer which was entered in **tb\_Answer** by setting its Text property to . Then set the properties of the **lb\_RightWrong** to: Text – Correct/Incorrect, Background Color – White, and Text Color to Black; otherwise set properties to: Text – Incorrect, Background Color – Red, and Text Color – White.
  - If this is the last question in the list then reset the **currentQindex** to 0.
  - Update the **currentQindex** and display the next question from the list in **lb\_QuestionLabel** with the appropriate picture by setting the Picture property of the image component.

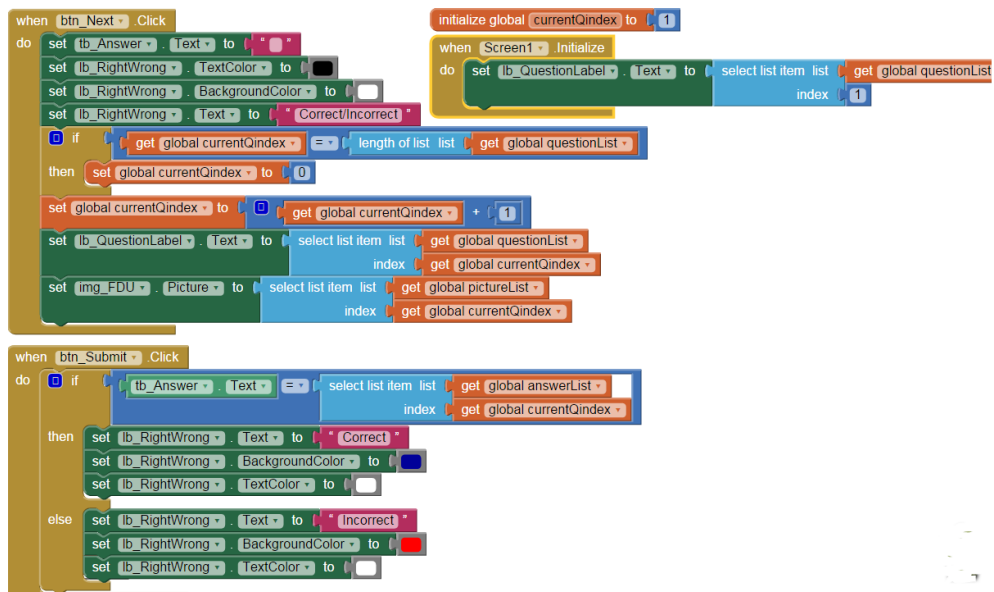


Fig 6.5 Blocks – Trivia

In this chapter we looked at how to create and populate **Lists**. We used ***pick a random item***, ***select list item*** with a particular index, and ***for each item in list*** to maneuver through a list. We also used a **clock** component to move the mole at regular intervals using a **procedure** to perform the movement. To ensure the mole appears in front of the hole we used the **Z-layering** approach for visual effect. We saw how to incorporate an image component to display different pictures by manipulating the image properties.

- Lists are used to create and manipulate different sets of values and/or elements.
- You can create empty lists in different ways.
- You can also query lists to get boolean results to find out if an item is in a list etc.
- Numerical values are returned when you ask for length of a list or position of an existing item in the list
- AI provides functions to add items to the end of a list, insert a list item at a specified position in a list, as well as append a list to another list.
- The clock fires timer events at regular intervals which can be used to trigger movement for example.
- The Z-layering approach is used for visual effect to make an object appear in front of another.
- AI provides functions such as for each item in list to maneuver through a list.



Please use the space below to summarize what you learnt in this chapter along with a summary of the Labs relevant to this chapter.

[illegible]

# BUILD ANDROID APPS WITH APP INVENTOR




## CHAPTER 6

[illegible]


## Chapter 7. Text and Colors

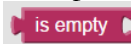




*“Colors, like features, follow the changes of the emotions.”– Pablo Picasso, a Spanish painter and sculptor.*

### ICON KEY


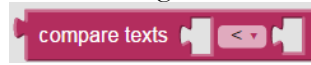
-  Valuable information
-  Keyboard exercise
-  Workbook review

The previous chapter introduced you to an important data structure the **List**, which makes infinity comprehensible and helps create order. In this chapter we will focus on **Text** and **Colors** which constitute core elements of an app.

 The **Text** and **Colors** blocks are available in the Built-in Blocks in the Blocks Editor and may represent the names of players in a game app or a palette of colors in a Paint app, etc. Most programming languages use text in the form of strings. AI considers any characters (letters, numbers, or other special characters) as a Text object and provides several functions for performing various operations with such objects.

The **Text** drawer in the Blocks pane under the Built-in tab provides different blocks for comparing, querying and manipulating strings. The  checks if the string contains any characters (including spaces) and returns a Boolean value true if the string length is 0 and false otherwise. To determine the length of a string you can use the  which gives the number of characters (including spaces) in the string. To remove leading or trailing spaces you can use . You can also convert strings to all uppercase or all lowercase using the  and .

### String Comparisons

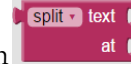
 Two strings can be compared lexicographically using  and selecting the appropriate operation from the drop down (= for equality, > for greater than, < for less than). The character order is taken into account for comparison. Computers use the American Standard Code for Information Interchange (**ASCII**, *pronounced ask -ee*) to represent the different characters on the keyboard for example. Each character is assigned a numerical value and comparison is based on these values. Upper case letters occur before the lowercase ones and have **lower** values. ASCII codes for A-Z are 65-90 contiguously and for a-z are 97-122. Thus, Apple<apple or apple>Apple would be true.



## Joining and Splitting Strings



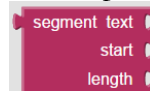
takes several inputs to make one single string. The mutator can be used to



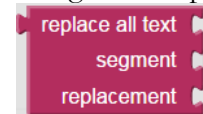
increase the number of inputs. String can be split in many ways with . The dropdown allows you to select different versions of split. The simplest one is **split** which divides *text* into pieces using *at* as the dividing point and produces a list of the result. **Split at first** divides the given *text* into two pieces using the location of the first occurrence of *at* as the dividing point, and returns a two-item list consisting of the piece *before* the dividing point and the piece *after* the dividing point. For example, if our text is Anna, Bob, Carl, Doug then setting at to , (our dividing point) the result would be a list with two items - Anna and Bob, Carl, Doug. There are other variations of **split** as well.

## Extracting and Replacing Strings

AI also provides blocks for extracting and replacing segments in a string.




is used to extract a substring from *text* starting at *start* position and



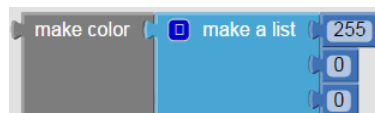
continuing for number of characters equal to *length*. results in a new string which is derived from *text* with all occurrences of segment replaced by *replacement*.

Many of the apps we create in the other chapters in this book use the Text blocks so we don't explain any apps in here.

## Colors

The **Colors** drawer in the Built-in tab of the Blocks pane can be used to define color in your app. You can make color or do the opposite to split color.  is a basic color block and has a color in the middle. Clicking on this displays a pop-up with a table of 70 colors from which you can choose a new color that will change the current color of your basic color block.

## Make and Split Color



takes several inputs and allows you to make color using three numbers (between 0 and 255) for the **Red (R)**, **Green (G)**, and **Blue (B)** components. You can generate a broad array of colors by changing the values

## Review

- ## Chapter and Lab Summary



## Chapter Summary

[illegible]

# BUILD ANDROID APPS WITH APP INVENTOR




## CHAPTER 7

[illegible]

## Chapter 8. Animation and Media

*“Animation is not the art of drawings that move but the art of movements that are drawn.”– Norman McLaren, a Scottish-born Canadian animator and film director.*

### ICON KEY

-  Valuable information
-  Keyboard exercise
-  Review

The previous chapter introduced you to Text and Colors which constitute core elements of an app. In this chapter we will focus on Animation as a medium of storytelling and visual entertainment. We will also look at Media and Storage components since most mobile device apps utilize various media resources for images, photos, sound and video as well data storage.

### Animation



AI uses sprite (**Ball** and **ImageSprite**) components and **Canvas** for creating animated, interactive, and lively apps. These are found in the Designer palette’s **Drawing and Animation** tab. A **Canvas** is a two-dimensional *touch-sensitive rectangular panel* on which users can draw and sprites can move. Sprites (**Ball** and **ImageSprite**) need a **Canvas** for moving and for different screen gestures such as dragging, touching, flinging, colliding etc. **ImageSprite** can take on *any* appearance from an image file, whereas **Ball** can only be a *round* sprite.

Sprites have three important properties:

- **Heading** – indicates the *angular degrees* above the positive x-axis with 0 degrees being towards the right, 90 to the top, 180 to the left and 270 to the bottom of the screen as shown in Fig 8.1.
- **Interval** – in *milliseconds* indicates how often the sprites position will be updated.
- **Speed** – indicates the *number of pixels* the sprite moves in every interval. For example, if a sprite has speed 5 and an interval of 10 milliseconds, then it moves 5 pixels every 10 milliseconds.

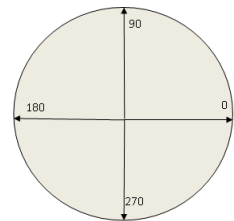


Fig 8.1 Heading degrees

Sprites generate events such as **Dragged**, **Flung**, **Touched**, **CollidedWith**, **EdgeReached** and have methods to **Bounce** and **MoveTo**. The Canvas also has the **Dragged**, **Flung**, **Touched** events and the **DrawCircle**, **DrawLine**, **Clear** and other methods.

## Media



To make an app come alive, AI uses Camera, Camcorder, ImagePicker, Player, Sound, SoundRecorder, etc. components, which are found in the Designer palette's **Media** tab. These are non-visible components and generate events and have methods. For example, the **Camera** component generates a **AfterPicture** event when the picture has been clicked and it has a **TakePicture** method to enable the device's camera to take a picture. All Android media formats for audio (MP3), image (JPEG, PNG), and video (MPEG-4) are supported.

## TinyDB



AI uses **TinyDB**, a non-visible component, to store data for an app. Apps are initialized each time they run so if an app sets the value of a variable and the user then quits the app, the value of that variable will not be remembered the next time the app is run. In contrast, TinyDB is a ***persistent*** data store for the app. The data stored there will be available each time the app is run. TinyDB is often used in a game app to save a high score. This high score is retrieved each time the game is played.

Data items are stored under tags. To store a data item, you need to specify a tag it should be stored under. Subsequently, to retrieve the stored data item you need to specify the given tag. Each device has only a single data store and thus multiple TinyDB components use the same data store. In order to get the effect of separate stores, you need to use different tags.

Each app has its own data store and you cannot use TinyDB to pass data between two different apps on the mobile device, although you can use it to share data between different screens of a multi-screen app.

## HungryMonkey App

We now create a HungryMonkey game app which uses animation and TinyDB components described above.



In this app we use both the **Ball** and a Monkey **ImageSprite** and a Banana **ImageSprite** on a **Canvas**. In this game the hungry monkey tries to capture a moving banana with a stone. Each time the monkey hits the target, the game score is updated. For this app, we will use the following components.

### Components

- Upload two image files (monkey and banana) and two mp3 sound files (game over and target hit) in the Media pane
- **ImageSprites**— to represent monkey and banana

- **Ball** – to represent the stone
- **Canvas** – a touch-sensitive surface to house the monkey, banana and stone.
- **Clock** – one to control the banana movement and one to control the game time
- **Labels**– to display the current game score and the high score on the device
- **Horizontal Arrangement** – to hold the score labels
- **TinyDB** – to hold the high score
- **Sound** – for game over and high score reached sounds



Start a **New Project** named **HungryMonkey**. The AI Designer will open. Set the Screen1's **Title** property to “HungryMonkey” You may also fill in the **AboutScreen** property to “Hungry Monkey Game” – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

### *Designer*

- Canvas **cnv\_HungryMonkey** (with Properties: Width – Fill Parent, Height – 300 pixels) containing a Ball (stone) and 2 ImageSprites (monkey and banana)
  - **bl\_Stone** (with Properties: Heading – 0, Interval – 100, PaintColor – Green, radius – 8, Speed – 0.0, X – 180, Y – 120, Z – 1)
  - **is\_banana** (with Properties: Heading – 0, Interval – 100, Picture: banana.png, Speed – 0.0, X – 170, Y – 20, Z – 1))
  - **is\_monkey** (with Properties: Heading – 0, Interval – 100, Picture: monkey.png, Speed – 0.0, X – 120, Y – 150, Z – 1)
- Horizontal Arrangement **HA\_Scores** containing 4 labels (set appropriate background and text colors of your choice for these labels)
  - Label **lb\_ScoreLabel** (with Properties: Text – Your Score: )
  - Label **lb\_Score** (with Properties: Text – 0)
  - Label **lb\_HighScoreLabel** (with Properties: Text – High Score: )
  - Label **lb\_HighScore** (with Properties: Text – 0)
- Button **btn\_Reset** (with Properties: BackgroundColor – Red, FontSize – 20, Text – Reset, TextColor – White)

- Label **lb\_GameOver** (with no Text)
- Clock **clk\_MoveTarget** from the Sensors tab of the Palette pane (with Properties: TimerAlwaysFires – checked, TimerEnbaled– checked and TimerInterval – 3000). This non-visible component fires every 3000 milliseconds or 3 seconds.
- Clock **clk\_GameOver** from the Sensors tab of the Palette pane (with Properties: TimerAlwaysFires – checked, TimerEnbaled– checked and TimerInterval – 60000). This non-visible component fires every 60000 milliseconds (every minute).
- TinyDB **TinyDB\_HighScore**, a non-visible component for persistent data storage on device
- Sounds **snd\_GameOver** and **snd\_HighScore** from the Media tab of the Palette pane. Note that these are non-visible component.

Fig 8.2 displays the Designer - Viewer and Components

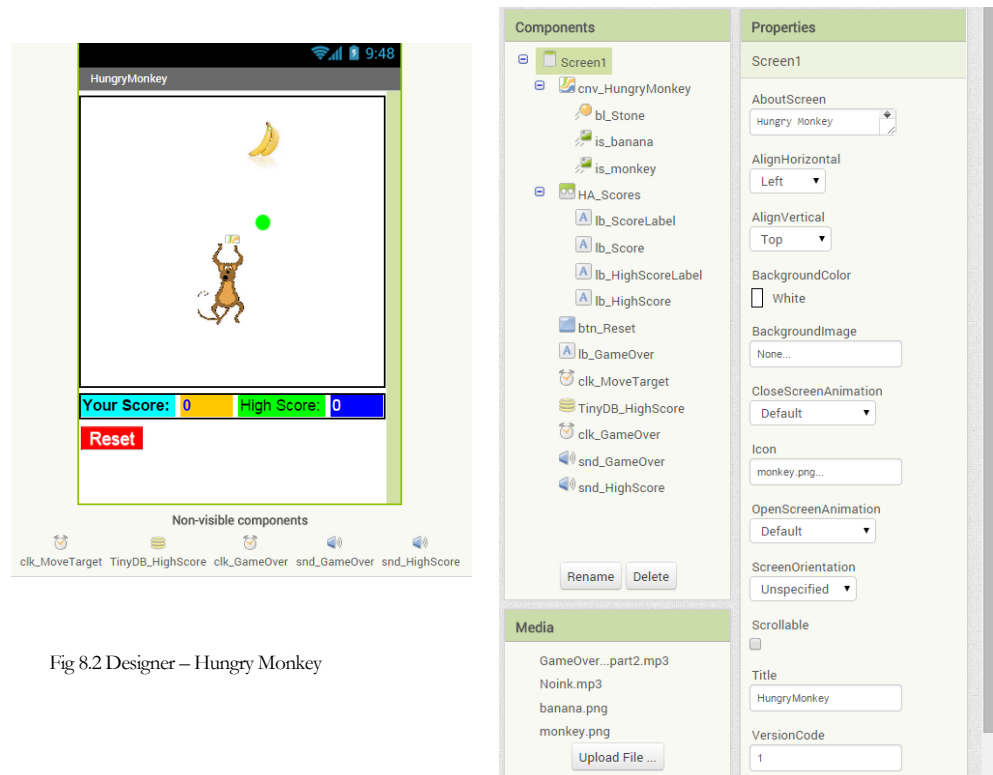

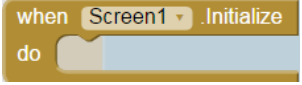


Fig 8.2 Designer – Hungry Monkey

Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

## Blocks

- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag  to workspace with name **hsScore** set to 0.

- Locate Screen1 in the Blocks pane and drag  to the workspace. Inside this add code block to do the following as shown in Fig 8.3 below:

- Make **bl\_Stone** (Property: Visible – False) invisible
- Get the value of High Score stored previously in the device (*persistent* data) from **TinyDB\_HighScore** using the **GetValue** function (set **tag** to **HS\_tag**, **valueIfTagNotThere** to 0) and set global variable **hsScore** to this value
- Display current High Score on screen via **lb\_HighScore**

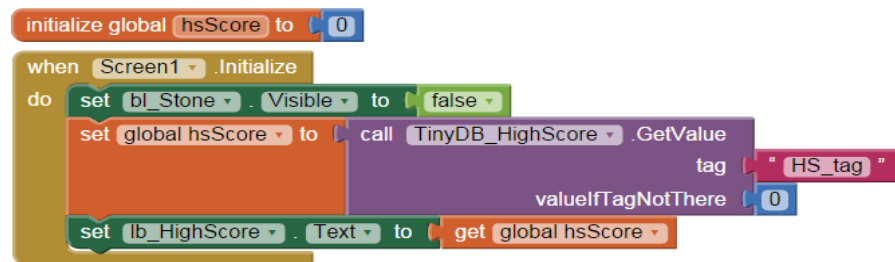


Fig 8.3 Initialize – HungryMonkey

- Locate imagesprite **is\_monkey** in the Blocks pane under Screen1-**cnv\_HungryMonkey** and drag two events - **Dragged** and **Touched** to workspace (as shown in Fig 8.4). When monkey is *dragged* you simply move it in a horizontal direction by setting *only* its **X** component to **currentX**. When it is *touched*, move the stone close to the monkey (with the stone's **x** and **y** properties set appropriately based on the width and position of the monkey). Also make the stone **Visible** and set its **Speed** to 5 and **Heading** to 90 (top vertical direction).

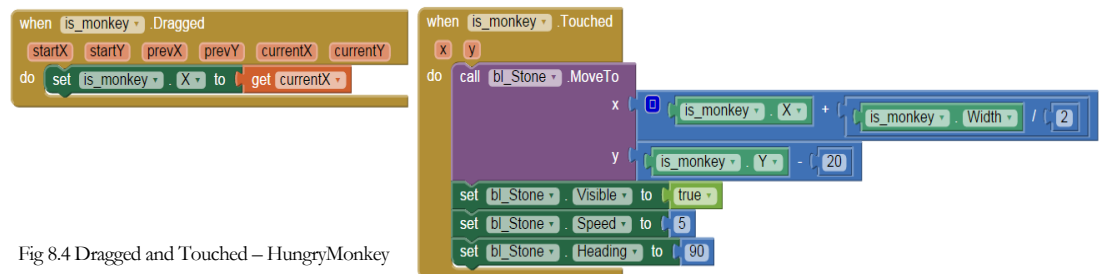


Fig 8.4 Dragged and Touched – HungryMonkey



- Locate ball **bl\_Stone** in the Blocks pane under Screen1-**cnv\_HungryMonkey** and drag two events - **CollidedWith** and **EdgeReached** to workspace (as shown in Fig 8.5). When stone *collides* (for example with the banana) make it invisible (as if monkey has eaten the banana) and update user's score by 1 point. Next we check to see if the user's current score is higher than the High Score already stored in the device. If it is a new High Score then play the sound for achieving the new High Score, update the persistent data with this new score using the **TinyDB StoreValue** function (using the same **tag** we used earlier in the **GetValue** function) and display this new High Score on screen via **lb\_HighScore**. Also, move the banana in the horizontal direction randomly. For this we use the **Math** function **random integer** from and set the limits from 0 to width of canvas – width of banana. When *edge is reached* (top edge in this case) simply make the stone invisible.

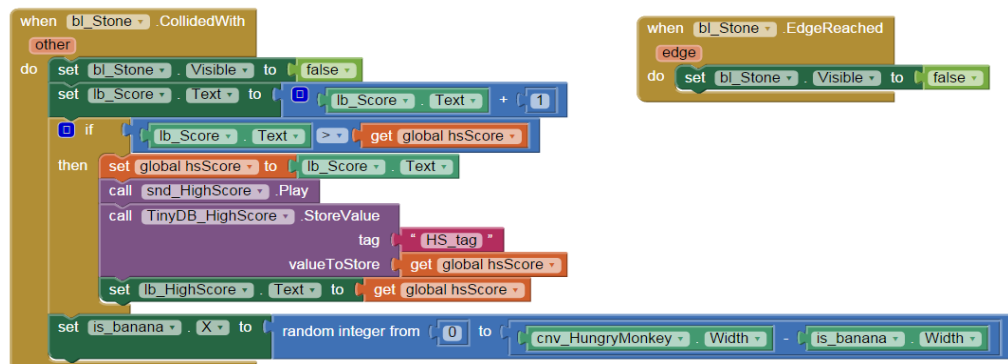


Fig 8.5 CollidedWith and EdgeReached – HungryMonkey

- Locate the Clock components **clk\_MoveTarget** and **clk\_GameOver** in the Blocks pane under Screen1 and drag their Timer events to the workspace as shown in Fig 8.6 below. When the **clk\_MoveTarget** Timer fires simply move the banana in the horizontal direction randomly as earlier. When the **clk\_GameOver** Timer fires, play the Game Over sound, display “Game Over” in **lb\_GameOver**, make the banana and stone invisible, **disable** the **clk\_MoveTarget** Timer and the imagesprite **is\_monkey** so that the game cannot be played anymore, until the Reset button is pressed (see the **btn\_Reset Click** event in Fig 8.6).

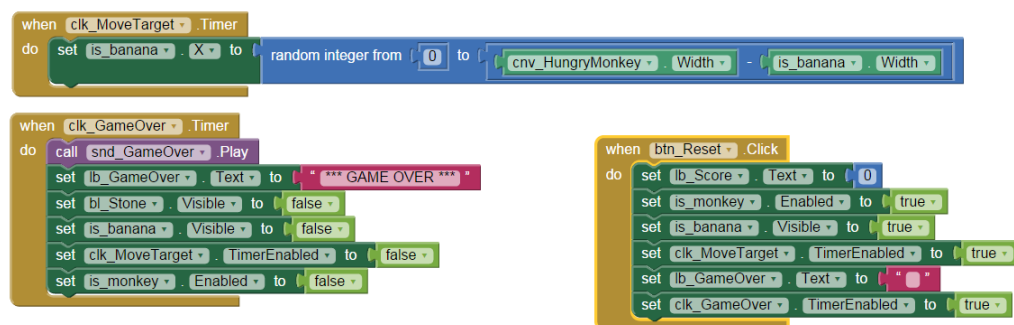


Fig 8.6 Clock Timers and Reset Button – HungryMonkey

## PaintPot\_PickColorScreen App

We now create a PaintPot\_PickColorScreen app to demonstrate how to work with multiple screens in an app. We will use the Colors Block and the process to make colors from the red, green, blue components we saw in Chapter 7.



AI has the ability to create apps with multiple screens, each having its own designer and blocks editor. Having too many screens makes your app too expensive as it uses a lot of computing resources. As a rule of thumb, try not to exceed ten screens in any app. We will create this app with two screens. The first screen is the main Paint Pot app (you may use Pink Background color for this screen to distinguish it from the other screen) and the second screen is used for picking a color of the user's choice. For this app, we will use the following components.

### Components

- Upload two image files (kitty.png and palette.png) in the Media pane
- Two Screens – Screen1 and Pick\_Color. Note that you cannot change the name of Screen1. For the second screen be careful about the name you pick since you cannot change it afterwards.
- Canvas to be used to paint something on the screen
- Buttons to wipe the screen clean and to pick color of choice for first screen. Buttons for TestColor, resetColor and Done on the second screen.



Start a **New Project** named **PaintPot\_PickColorScreen**. The AI Designer will open. Set the Screen1's **Title** property to "Paint Pot." You may also fill in the **AboutScreen** property to "Pick a color and paint the cat!" – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

- Canvas **cnv\_Kitty** (with Properties: BackgroundImage – kitty.png, Width – Fill Parent, Height – Automatic, PaintColor – Red)
- Two buttons **btn\_Wipe** (with Properties: BackgroundColor – Magenta, FontBold, FontSize – 20, Text – Wipe, Width – Fill Parent, Height – Automatic) and **btn\_PickColor** (with Properties: FontBold, FontSize – 16, Image – palette.png, Text – Pick A Color, TextColor – Blue, Width – Automatic, Height – Automatic) as shown in Fig 8.7

Click **Add Screen ...** to **add the second screen** – set name to Pick\_Color. On the second screen add the following components

- Horizontal Arrangement **HA\_ColorCanvas** containing a Vertical Arrangement **VA\_RGB** and a label **lb\_MakeYourOwnColor** (with Properties: BackgroundColor – LightGray, FontSize – 20, Text – about 25 blanks followed by the words Make Your Own Color, TextColor – Blue, Width – Fill Parent, Height – Fill Parent). Inside the **VA\_RGB** add 3 Horizontal Arrangements **HA\_Red**, **HA\_Green** and **HA\_Blue**. Add a label **lb\_RedLabel** and Textbox **tb\_Red** (with Properties: Hint – Enter 0-255 ) to the **HA\_Red** and repeat for **HA\_Green** and **HA\_Blue** as shown in Fig 8.8
- Horizontal Arrangement **HA\_TestReset** containing two buttons **btn\_TestColor** (with Properties: Text – Test Color, Width – Fill Parent, Height – Automatic) and **btn\_ResetColor** (with Properties: Text – Reset Color, Width – Fill Parent, Height – Automatic) with a label **lb\_ColorSample** (with Properties: FontSize – 20, Text – about 25 blanks, Width – 50 pixels, Height – Automatic) in between
- Button **btn\_Done** (with Properties: Text – Done, Width – Automatic, Height – Automatic)

Fig 8.7 and 8.8 display the Designer - Viewer and Components for Screen1 and Screen2 respectively.

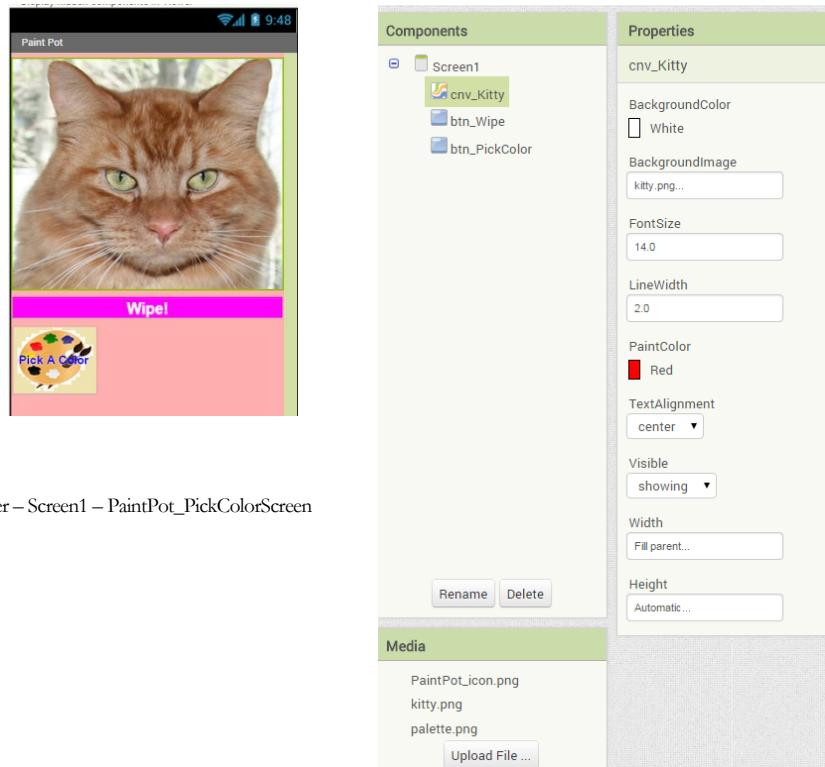


Fig 8.7 Designer – Screen1 – PaintPot\_PickColorScreen

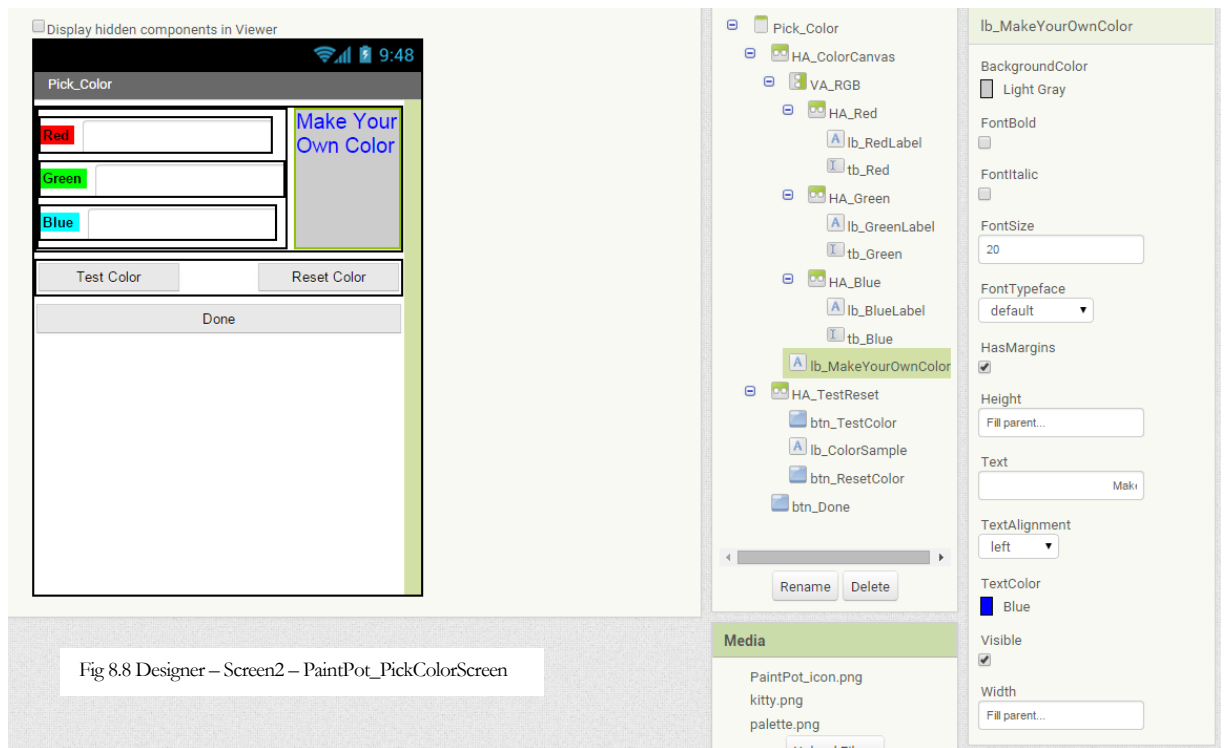


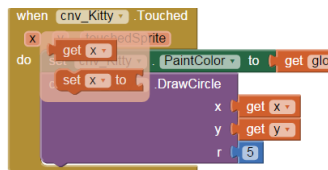


Fig 8.8 Designer – Screen2 – PaintPot\_PickColorScreen

Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace screen by screen. Let us begin with the first screen. The blocks for Screen1 are shown in Fig 8.9

### Blocks

- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag **initialize global name to** to workspace with name **currentColor** set to  from the Colors drawer in the Blocks pane under the Built-in tab.
- Locate Screen1 in the Blocks pane and drag **when Screen1 Initialize** to the workspace and set **cnv\_Kitty PaintColor** to  to
- Locate canvas **cnv\_Kitty** in the Blocks pane under Screen1 and drag two events - **Dragged** and **Touched** to workspace. For both events the first thing you do is set **cnv\_Kitty PaintColor** to **currentColor** (this color will be chosen on the second screen). For the **Touched** event draw a circle of radius 5 at the point (x, y) where the canvas was touched and for the **Dragged** draw a line from previous to current. Hover near the x and



select get x  
currentX, currentY.

and similarly for y, prevX, prevY,

The next two steps are related to the two screens (we are still in the Blocks Editor of Screen1)

- Locate the **btn\_PickColor** in the Blocks pane under Screen1, drag the click event to the workspace and add the **open another screen screenName** found in the Control drawer in the Blocks pane under the Built-in tab. Make sure to provide the correct screen name for the second screen – **Pick\_Color**.

- Locate Screen1 in the Blocks pane and drag to the workspace and set **currentColor** to the result (need to hover on result and then drag get result) returned from the second screen as shown in Fig 8.9.

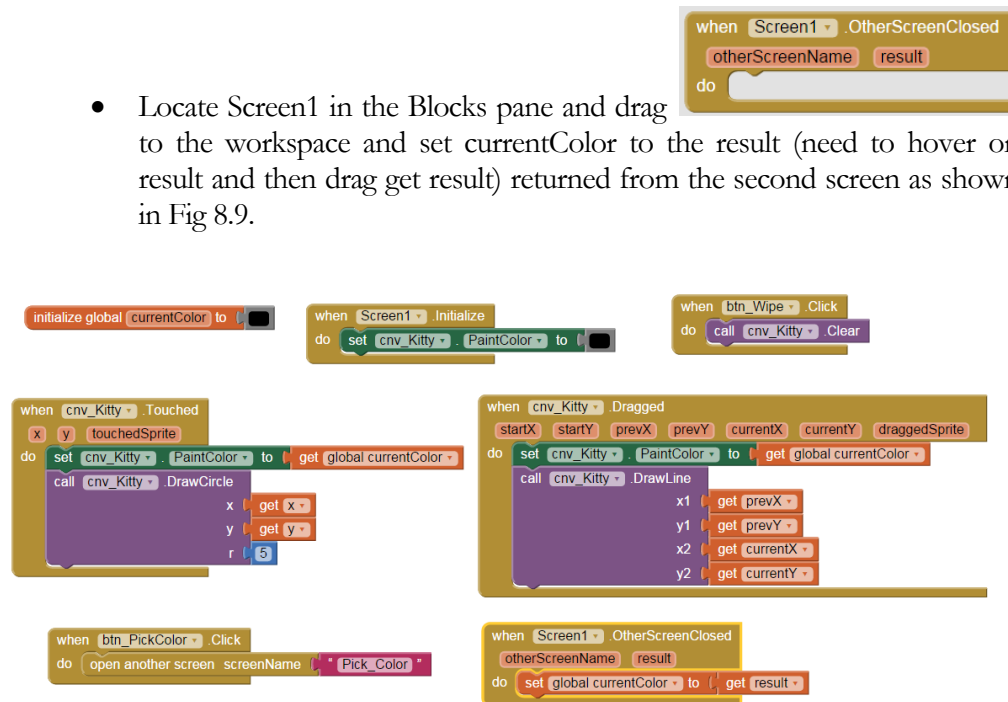



Fig 8.9 Blocks – Screen1 – PaintPot\_PickColorScreen

Now, let us work on the second screen **Pick\_Color**. The blocks for Screen2 are shown in Fig 8.10

## Blocks Screen2

- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag **initialize global name to** to workspace with name **currentColor** set to  from the Colors drawer in the Blocks pane under the Built-in tab.

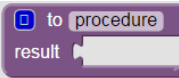

- Locate **Pick\_Color** (name of the second screen) in the Blocks pane and drag the initialize event to the workspace and set **lb\_ColorSample** BackgroundColor to **currentColor**.
- Locate the **Procedures** drawer in the Blocks pane under the Built-in tab and drag two  to the workspace and name the procedures **checkColor** (with one input - **colorInput**) and **limitRange** (with three inputs – input, low, high). Use the mutator to add required number of inputs to the two procedures. The first procedure takes in **colorInput** and calls the **limitRange** procedure with this **colorInput** and low of 0 and high of 255 to make sure its value is between 0 and 255. Of course this same work can easily be accomplished with a single procedure but we wanted to show how easy it is to have one procedure call another and maybe the **limitRange** procedure code could be reused elsewhere. the **limitRange** procedure is a classic code for limiting a value between low and high. If the **colorInput** exceeds 255 it will be clipped to 255 (the high value), and if the **colorInput** is less than 0 then it will set to 0 (the low value). We will call the **checkColor** procedure when we make the color as per the user-entered red, green and blue components.
- Locate the **btn\_TestColor** in the Blocks pane under **Pick\_Color**, drag the click event to the workspace and add the following code
  - Use **makeColor** with a list of red, green, blue components (already checked to be between 0 and 255) to set the **currentColor**.
  - Set **lb\_ColorSample** BackgroundColor to **currentColor** so that the user can see a little swatch of the color.
  - On the **cnv\_ColorCircleSample** draw a circle of radius 10 at x=270 and y=35 so that the user can see this color in a larger sample.
- Locate the **btn\_ResetColor** in the Blocks pane under **Pick\_Color**, drag the click event to the workspace and clear out Text in **tb\_Red** and set the Hint to Enter 0-255. Repeat for the other two textboxes.
- Locate the **btn\_Done** in the Blocks pane under **Pick\_Color**, drag the click event to the workspace and  to close the second screen and pass the **currentColor** to the first screen. Note that this result is used by Screen1 when the second screen is closed (refer to the blocks for Screen1 event when other screen closed).

Fig 8.10 shows the blocks for the second screen **Pick\_Color**.

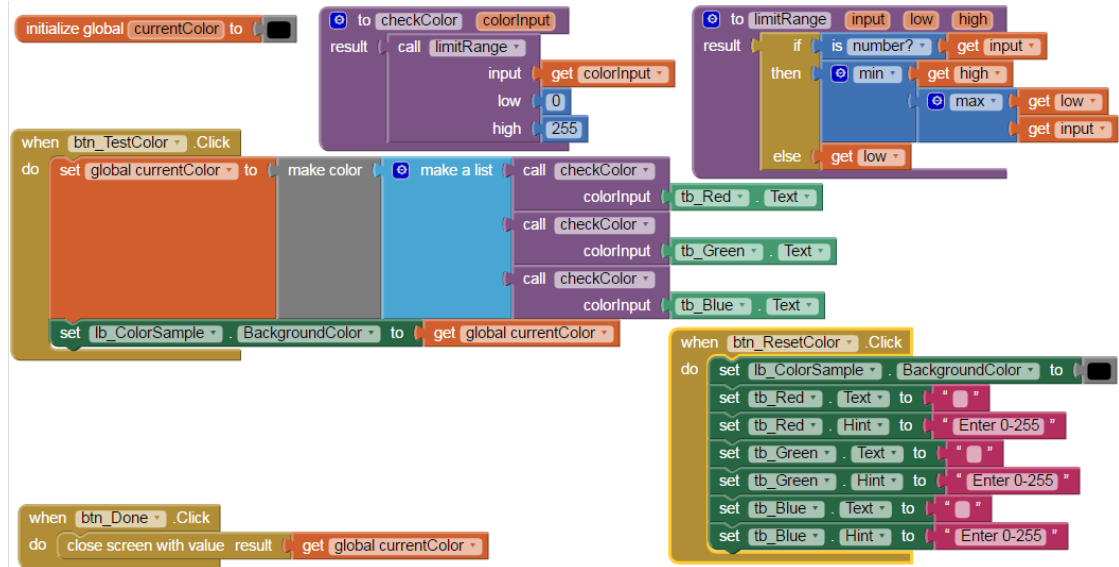


Fig 8.10 Blocks – Screen2 – PaintPot\_PickColorScreen

## PaintPot\_Camera App

We now create a PaintPot\_Camera app to demonstrate how to work with the camera on your device. We will also use a Slider component - a progress bar that adds a draggable thumb, which you can drag left or right to set the slider thumb position.



Camera is a non-visible component, found in the Media tab, takes a picture using the device's camera. After the picture is taken, the AfterPicture event triggers and the image can be used to set the background image of the canvas in our app. The Slider is a visible component found in the UserInterface tab and as the Slider thumb is dragged, it will trigger the PositionChanged event, reporting the position of the Slider thumb. The reported position is used to dynamically update another component attribute, such as the dotSize in this App. For this app, we will use the following components.

### Components

- Upload two image files (kitty.png and camera.png) in the Media pane
- **Horizontal Arrangements** – one to hold Red, Green and Blue buttons for selecting a color and another to hold the TakePicture button and the slider.
- **Canvas** to hold a picture (kitty.png or the one taken with the device's camera)
- **Labels**– to display the dot size
- **Buttons** to TakePicture and Wipe the canvas clean

- **Camera** component



Start a **New Project** named **PaintPot\_Camera**. The AI Designer will open. Set the Screen1's **Title** property to "Paint Pot." You may also fill in the **AboutScreen** property to "Pick a color and paint the cat or click a picture to paint" – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.


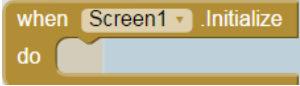

### *Designer*

- Horizontal Arrangement **HA\_Buttons** (with Property: Width– Fill Parent, Height – Automatic) to hold 3 buttons **btn\_Red** (with Property: BackgroundColor – Red, Text – Red, TextColor – Yellow, Width – Fill Parent, Height – Automatic), **btn\_Blue** (with Property: BackgroundColor – Blue, Text – Blue, TextColor – White, Width – Fill Parent, Height – Automatic), **btn\_Green** (with Property: BackgroundColor – Green, TextColor – White, Text – Black, Width – Fill Parent, Height – Automatic), as shown in Fig 8.11
- Canvas **cnv\_kitty** (Property: BackgroundImage – kitty.png, Width – Fill Parent, Height – 250 pixels)
- Button **btn\_Wipe** (with Properties: Text – Wipe, Width – Fill Parent, Height – Automatic)
- Horizontal Arrangement **HA\_Camera\_Slider** (with Property: Width– Fill Parent, Height – Automatic),to hold a buttons **btn\_TakePicture** (with Property: Image – camera.png, Text – Click Picture), a label **lb\_DotSize** (Property: Text – Dot Size: ) and a Slider **Slider\_Size**(with Property: ColorLeft – Blue, ColorRight – Gray, MaxValue – 20, MinValue – 2, ThumbPosition – 5, Width – Fill Parent)
- Non-visible component Camera1 from the Media tab

Fig 8.11 displays the Designer - Viewer and Components

Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

### *Blocks*

- Locate the **Variables** drawer in the Blocks pane under the Built-in tab and drag  to workspace with name **dotSize** set to 5.
- Locate Screen1 in the Blocks pane and drag  to the workspace and set **cnv\_kitty** PaintColor to 



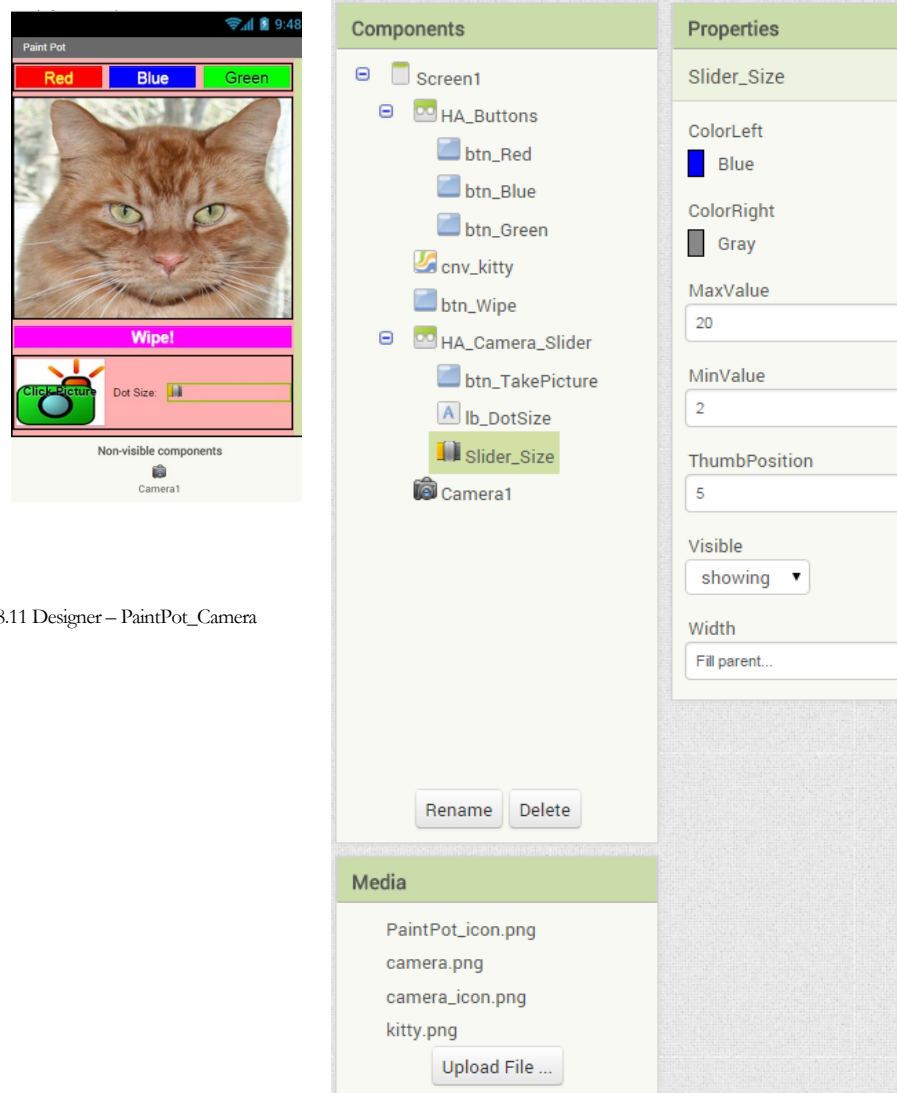



Fig 8.11 Designer – PaintPot\_Camera

- Locate the **btn\_Wipe** in the Blocks pane under Screen1, drag the click event to the workspace to clear the canvas.
- Locate the **btn\_Red** in the Blocks pane under Screen1, drag the click event to the workspace to set **cnv\_Kitty** PaintColor to . Repeat for **btn\_Green** and **btn\_Blue**.
- Locate canvas **cnv\_kitty** in the Blocks pane under Screen1 and drag two events - **Dragged** and **Touched** to workspace. For the **Touched** event draw a circle of radius dotSize at the point (x, y) where the canvas was touched and for the **Dragged** draw a line from previous to current.

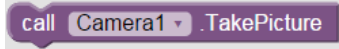
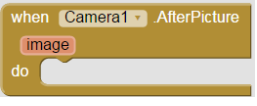

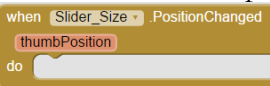

- Locate the **btn\_TakePicture** in the Blocks pane under Screen1, drag the click event to the workspace to  from the Camera1 in the Blocks pane under Screen1 HA\_Camera\_Slider.
- For the Camera1 in the Blocks pane under Screen1 HA\_Camera\_Slider  
  
drag the  and set **cnv\_Kitty** BackgroundImage to get image.
- For the Slider **Slider\_Size** in the Blocks pane under Screen1  
  
HA\_Camera\_Slider drag the  and set dotSize to thumbPosition.

Fig 8.12 shows the blocks for this app.

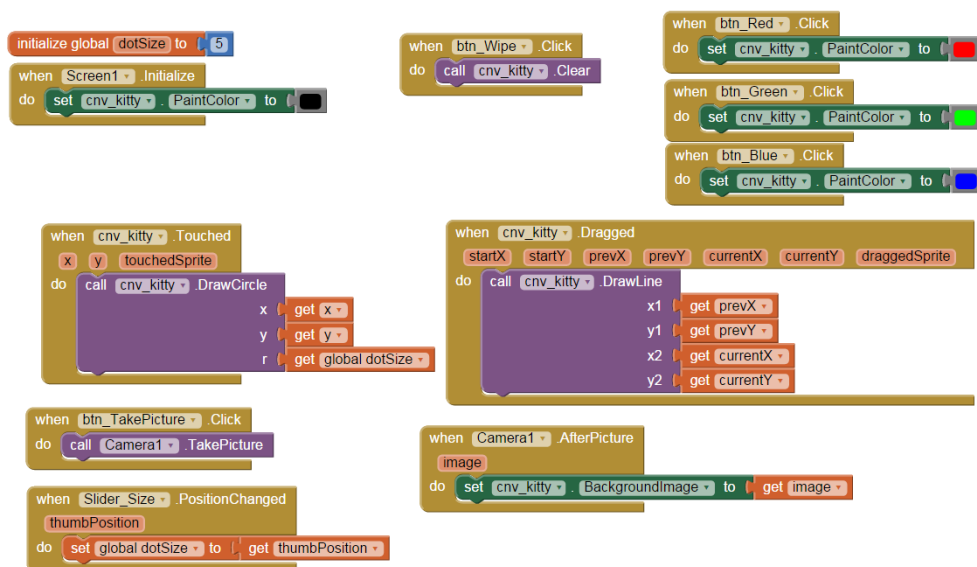


Fig 8.12 Blocks – PaintPot\_Camera

In this chapter we looked at how to use animation and media in our apps. TinyDB allows us to use the device to store persistent data. Having the ability to use multiple screens allows your app to look uncluttered. In the next chapter we will explore Sensors.



## Review

- AI uses Animation as a medium of storytelling and visual entertainment.
- Most mobile device apps utilize various media resources for images, photos, sound and video as well data storage.

- AI uses Ball, ImageSprite and Canvas for creating animated, interactive, and lively apps.
- ImageSprite can take on any appearance from an image file, whereas Ball can only be a round sprite.
- Canvas is used for ball and image sprite movement. It is also used for different screen gestures such as dragging, touching, flinging, colliding etc.
- AI provides Media components such as Camera, Camcorder, ImagePicker, Player, Sound and SoundRecorder.
- TinyDB is a persistent data store for an app and is useful for retaining data between runs of the app on a device.
- Multiple screens allow your app to look uncluttered and are easy to implement.



## Chapter and Lab Summary

Please use the space below to summarize what you learnt in this chapter along with a summary of the Labs relevant to this chapter.




[illegible]

[illegible]

## Chapter 9. Sensors and Connectivity

*“The great art of life is sensation, to feel that we exist, even in pain.”—Lord Byron, an English poet.*

### ICON KEY

-  Valuable information
-  Keyboard exercise
-  Review

The previous chapter introduced you to Animation and Media which make an app come alive. In this chapter we will focus on Sensor and Connectivity components which are an integral part of most mobile devices.

Modern Android mobile devices are equipped with a range of built-in sensors which can measure motion, orientation, and various environmental conditions. These sensors (in the Sensors tab of the palette in the AI designer) provide data to monitor device movement or position. For example, a game app may need to track user gestures and motions such as tilt or shake, or a travel app might use the geomagnetic field sensor and accelerometer to report a compass bearing.

### AccelerometerSensor



AI uses Accelerometer sensor to detect shaking and measure acceleration on the device. This non-visible component generates the AccelerationChanged and Shaking events and has several properties including:

- **xAccel** 0 when phone is at rest on a flat surface, positive when phone is tilted to the right, negative if tilted to the left
- **yAccel** 0 when phone is at rest on a flat surface, positive when phone's bottom is raised, negative when phone's top is raised
- **zAccel** 0 when device is perpendicular to the ground, negative 9.8 when the device is at rest parallel to the ground with display facing up and positive 9.8 with display facing down - note that the earth's gravity is  $9.8 \text{ m/s}^2$  (meters per second square).

### BarcodeScannerSensor



AI provides a BarcodeScanner, a non-visible component, to read a barcode. This component provides a DoScan method that uses the device's camera to read a barcode. When the scan is complete, this method generates the AfterScan event which provides a text result.

## Clock



This is a non-visible component that provides the instant in time using the devices' internal clock. It fires a timer at regular intervals and performs time calculations, manipulations, and conversions. When the timer goes off it generates the `Timer` event. It has methods to manipulate and format date and time. It has the following properties:

- **TimerAlwaysFires** when set to true, the timer will fire even when the app is not showing on the screen
- **TimerEnabled** the timer will fire when set to true
- **TimerInterval** Interval in milliseconds (ms – thousandth of a second) between timer events

## LocationSensor



This non-visible component provides location information, such as longitude, latitude, altitude (if supported by the device), and address. This has methods to perform “geocoding” – converting an address to latitude and longitude. When a new location for the device is detected, a `LocationChanged` event is generated. This component has several properties including: `Altitude`, `CurrentAddress`, `Latitude` and `Longitude`. The `DistanceInterval` and `TimeInterval` properties specify the minimum distance (in meters) and the minimum time (in milliseconds) the sensor will try to use for sending out location updates.

## NearField



This non-visible component provides Near Field Communications (NFC) capabilities (for example to transfer data between devices when they are in proximity) and supports the reading and writing of text tags (if supported by the device). In order to read and write text tags, the component must have its `ReadMode` property set to `True` or `False` respectively. This generates the `TagRead` event when a new tag is detected.

## OrientationSensor



An orientation sensor is a non-visible component that determines a device's spatial orientation and reports the roll, pitch and azimuth. This component generates the `OrientationChanged` event and has several properties including:

- **Roll** – roll angle of the device in degrees – 0 when the device is level, increasing to 90 as the device is tilted up onto its left side, and decreasing to -90 when the device is tilted up onto its right side.

- **Pitch** – the pitch angle of the device in degrees – 0 when the device is level, increasing to 90 as the device is tilted so its top is pointing down, then decreasing to 0 as it gets turned over. Similarly, as the device is tilted so its bottom points down, pitch decreases to -90, and then increases to 0 as it gets turned all the way over.
- **Azimuth** – the azimuth angle of the device in degrees – 0 when the top of the device is pointing north, 90 when it is pointing east, 180 when it is pointing south, 270 when it is pointing west, etc.

## Connectivity



Mobile devices are used for a variety of activities such as performing a web search, connecting via Bluetooth® (without wires) to peripheral devices – speakers and headphones, surfing the web, starting apps such as the camera app, etc. AI provides several components to do this. An **ActivityStarter** is a non-visible component that can launch an activity using the **StartActivity** method. For Bluetooth connectivity, AI provides the **BluetoothClient** and **BluetoothServer** components. The **Web** component provides functions for web requests.

## LocationMap App

We now create a **LocationMap** app which uses some of the sensor and connectivity components described above.



### *Components*

In this app the device can get its current location and display the latitude, longitude and address and also the map of this location. The user can also enter any address to view it on the map. For this app, we will use the following components.

- **Labels**– to display the current address, latitude, longitude, etc.
- **Buttons** – to get current location, display current location on map and display entered address on map.
- **TextBox** – to enable user to enter any address
- **Horizontal** and **Vertical Arrangements**– to hold the buttons and labels
- **LocationSensor** – for location information - longitude, latitude, address
- **TextToSpeech** – to announce the location when it changes
- **ActivityStarter** – to start the map view activity
- **Notifier** – to display alert dialog when user tries to map a blank address



Start a **New Project** named **LocationMap**. The AI Designer will open. Set the Screen1's **Title** property to "LocationMap" You may also fill in the **AboutScreen** property to "Location Sensor and Activity Starter App" – this will be displayed when user clicks on **About this Application** when your app runs.

Using the AI Designer add the following components to Screen1 and change some of the properties.

### *Designer*

- Labels **lb\_Lat**, **lb\_Long** and **lb\_Addr** (clear out Text)
- Horizontal Arrangement **HA\_Location\_Map** (with Properties: Width – Fill parent, Height – Automatic) containing 2 buttons (set appropriate background and text colors of your choice)
  - Button **btn\_GetLocation** (with Properties: Text – Get Location)
  - Button **btn\_ShowLocationOnMap** (with Properties: Text – Show Location On Map)
- Label **lb\_WaitLabel** (with Properties: Text – Please wait! Your location is being determined, Visible – Hidden). Note that this label is initially hidden and will be made visible in the code later.
- Vertical Arrangement **VA\_Address\_Map** (with Properties: Width – Fill parent, Height – Automatic) containing label, textbox and button (set appropriate background and text colors of your choice)
  - Label **lb\_EnterAddress** (with Properties: Text – Enter Address :)
  - textbox **tb\_EnterAddress** (with Properties: Hint – Enter Address here, blank out the Text)
  - Button **btn\_Map** (with Properties: Text – Show Address On Map)
- Location Sensor **LocationSensor1** (with Properties: TimeInterval – 60000). Note that this time interval represents a minute (60000 milliseconds = 60 seconds = 1minute)
- Text To Speech component **TextToSpeech1** (with default properties)
- Activity Starter **ActivityStarter1** (without any properties, we will fill the properties in the code)
- Notifier **Notifier1** (with default properties)

Fig 9.1 displays the Designer - Viewer and Components



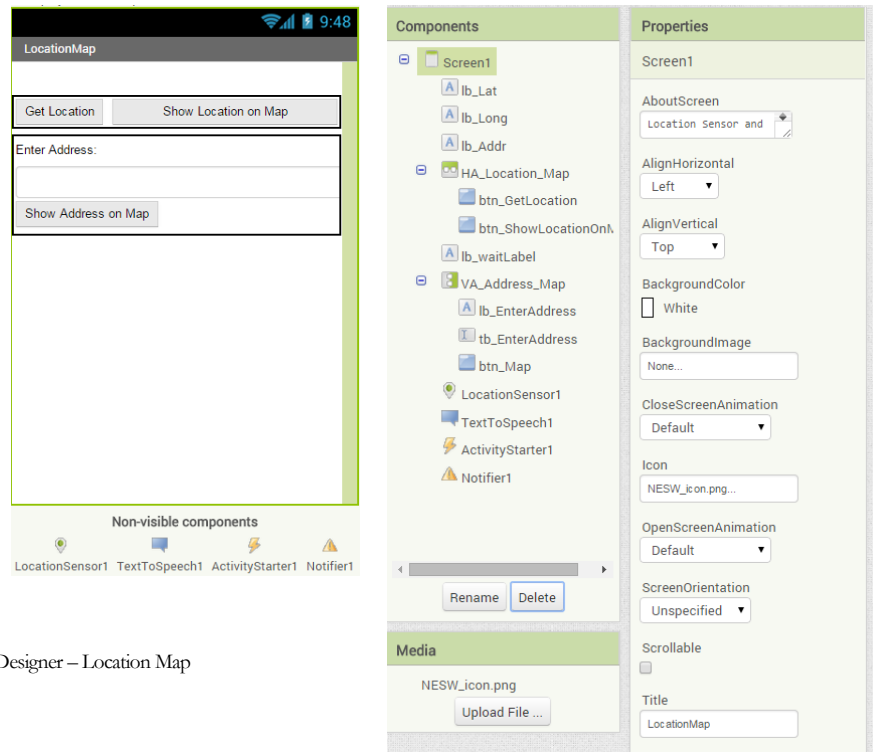
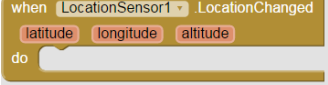


Fig 9.1 Designer – Location Map

Click on Blocks in the upper right of the AI Designer and add the following code blocks in the Blocks Editor workspace.

### *Blocks*

- Locate the **btn\_GetLocation** under **HA\_Location\_Map** in the Blocks pane under Screen1 tab and drag the click event to workspace. Enable **LocationSensor1** and make the **lb\_waitLabel** visible as shown in Fig 9.2.

- For LocationSensor1 under the Screen1 tab in the Blocks pane drag the  to the workspace and set the latitude, longitude and address labels to the values obtained from the Location Sensor as shown in Fig. 9.2. Announce location has changed and the new

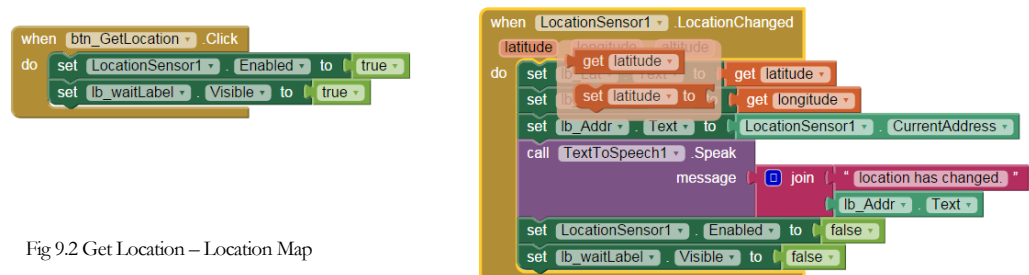


Fig 9.2 Get Location – Location Map

address via TextToSpeech. Disable Location Sensor and hide wait label.

- Locate the **btn\_ShowLocationOnMap** under **HA\_Location\_Map** in the Blocks pane under Screen1 tab and drag the click event to workspace. Locate ActivityStarter1 under Screen1 tab in the Blocks pane and drag **set ActivityStarter1 . Action to** and **set ActivityStarter1 . DataUri to** to the workspace as shown in Fig 9.3. This is where we specify the activity to view the location address on a map. Repeat the same for the **btn\_Map** under **VA\_Address\_Map** in the Blocks pane under Screen1 tab. For this we also add a check to see if the user has entered an address in the TextBox **tb\_EnterAddress**. If the textbox is empty then we notify the user via a dialog using Notifier1.

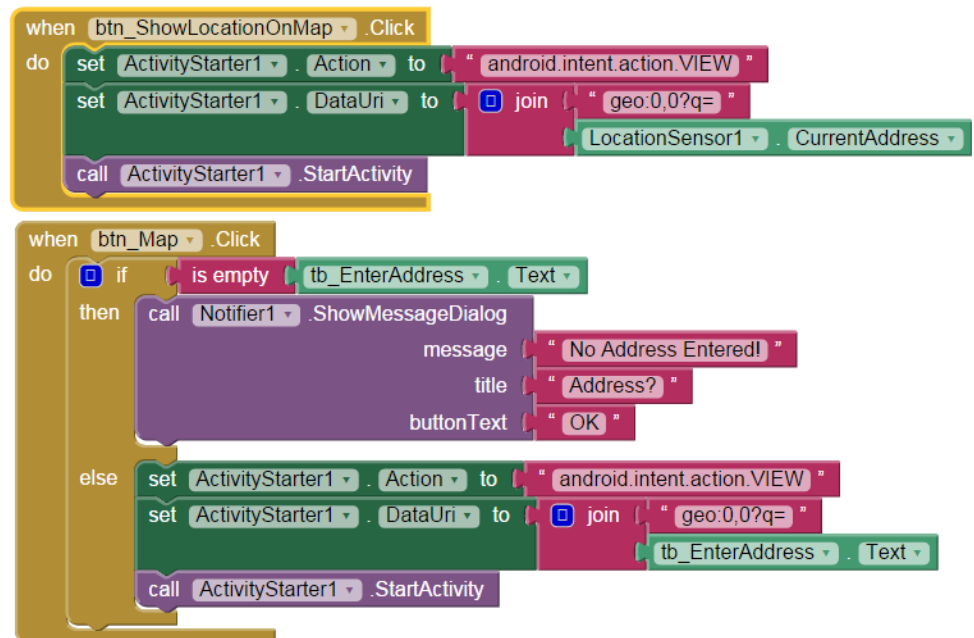


Fig 9.3 Show on Map – Location Map

In this chapter we looked at how to use sensor and connectivity components in our apps. Other real-world apps can be easily made using the other sensors and connectivity components we described in this chapter.

## StockQuote App

We now create a StockQuote app which uses the Web component to look up a stock price based on a stock symbol a user enters. Use the designer to create the user interface which has a textbox (to input the stock symbol), a button (to get the stock quote), a label (to display the stock price), and a Web component from the Connectivity group. When the Get Quote button is pressed set the Web Uniform

Resource Locator (URL) to a service provider's web service to request a quote – for example use Yahoo finance <https://finance.yahoo.com/quote/SIRI> as a provider to get a stock quote for the Sirius XM Holdings Inc. (stock symbol SIRI) and then perform a get on the web component. When the response to the web request arrives, a GotText event is raised on the Web component. In this event you can display the responseContent in the label for a successful responseCode (200). Otherwise display an error.

## FindConcert App

Another fun app is the FindConcert App which uses the WebViewer component to find a concert for a particular artist in a particular region. You specify a name of the artist, the location (city or state for example) where/when you want to see their performance, a year for example, etc. The app uses a webviewer component to view concert dates and locations of the artist you searched for in your neighborhood. It uses a search engine on songkick.com site with a query such as: <https://www.songkick.com/search?type=initial&query=> with the artist name and other details appended.



## Review

- Sensor and Connectivity components are an integral part of most mobile devices.
- AI uses Accelerometer sensor to detect shaking and measure acceleration on a mobile device.
- AI provides a BarcodeScanner component to read a barcode.
- Clock provides the instant in time using a devices' internal clock.
- LocationSensor provides location information, such as longitude, latitude, and address.
- AI provides OrientationSensor to determine a device's spatial orientation and reports the roll, pitch and azimuth.
- A variety of activities such as performing a web search, connecting via Bluetooth to peripheral devices - speakers, headphones, surfing the web, starting apps such as the camera app are available in AI.



## Chapter and Lab Summary

Please use the space below to summarize what you learnt in this chapter along with a summary of the Labs relevant to this chapter.

[illegible]




[illegible]

## Chapter 10. Packaging Apps

*“There are only two mistakes one can make along the road to truth; not going all the way, and not starting.” - Buddha, a sage who founded Buddhism.*

*Every individual has his own style, his own way of presenting himself on and off the field.” – Sachin Tendulkar, an Indian cricketer.*

### ICON KEY

-  Valuable information
-  Keyboard exercise
-  Review

The previous chapters introduced you to develop apps for mobile devices using the App Inventor. In this chapter you will see how to package and present your apps.

After you create your app you may want to share it with others. Sharing can done in different ways. One way to share your app is in the source code form. Another way is the executable form. Distributing your app via the Google Play store is yet another option.

### Sharing the Source Code



In a classroom setting your instructors/teachers/graders etc. may want you to share the source code of your app so that they can see your design and blocks code. AI uses the project.**aia** file to share the source code. These **.aia** files can only be opened or viewed in AI.

Open your app in AI and locate the **Projects** drop down on the top. From this, click on the “**Export selected project (.aia) to my computer**” as shown in Fig 10.1. This will download **ProjectName.aia** to your computer (check the Downloads folder). You can also do the export from the **Projects** drop down - **My Projects** tab making sure to check ☒ the project you want to export as shown in Fig 10.2. You cannot check ☒ multiple projects at the same time. You need to export one by one.

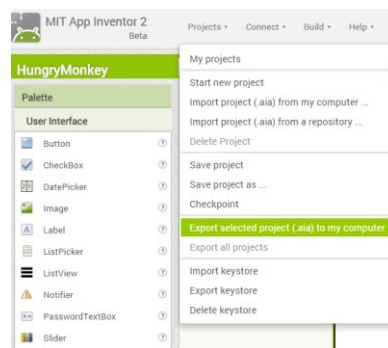


Fig 10.1 Export aia from Projects drop down

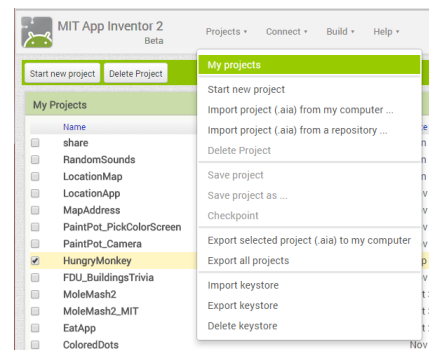


Fig 10.2 Export aia from My projects

In AI, to open a project.aia file that someone has shared with you, you can locate the **Projects** drop down on the top and click on the “**Import project (.aia) from my computer**” and choose the file. This will open

## Sharing the Executable Form



You do not have to share the source code with others who simply want to use your app and in that case you would share the executable form of your app via the **.apk** file, which they can download and install on their device.

Open your app in AI and locate the **Build** drop down on the top. Then click on “**App (save .apk to my computer)**” as shown in Fig 10.3. This starts the build process and you will see a progress bar in the pop-up alert box as shown in Fig 10.4.

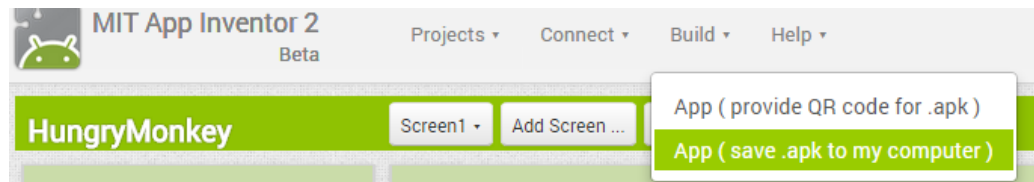


Fig 10.3 Build App

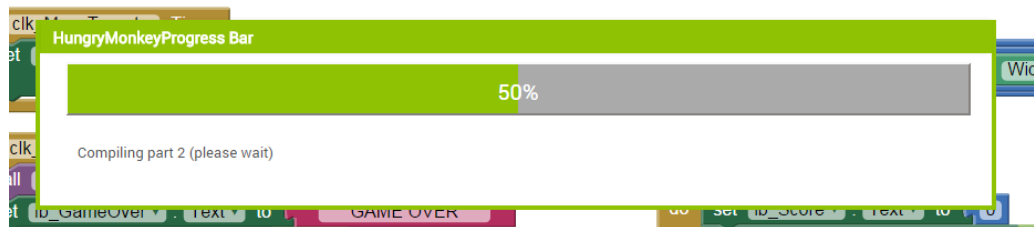


Fig 10.4 Build App – Progress Bar

Once the build completes you can share the **.apk** file by email or by uploading it to a website from where people can download the **.apk** file onto their device. Anyone installing your app (the **.apk** file) will need to change the setting on their device to allow installation of non-market applications. You can find this setting under **Settings - Applications** . Check the box next to "Unknown Sources". For some devices you may find this under **Settings - Security** or **Settings - Security & Screen Lock** and then check the box next to **Unknown Sources** and confirm your choice.

The other option on the **Build** drop down - App (provide QR code for .apk) - generates a QR code for the **.apk** as shown in Fig 10.5. This code can be scanned by anyone who needs to install your app on their device (using any QR code scanner on their device). However, this code is only valid for 2 hours from the time it was generated.

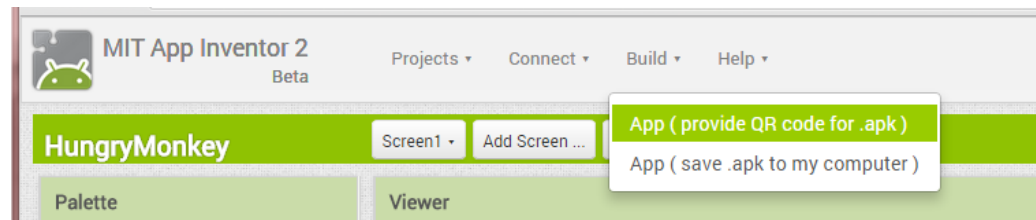


Fig 10.5 Build App – QR code

## App Distribution via Google Play



AI apps can also be published via Google Play which requires a registration fee. Publishing guidelines and details for developers are found at Google Play. Before you generate the .apk file, make sure your app has the **VersionCode** and **VersionName** fields populated as in Fig 10.6. This can be done in the AI designer under the properties panel for the Screen1 component.

**VersionCode** (an integer value, invisible to Google Play Store users) defaults to 1 and should be increased by one with every *successive* change whether it is a *major* change or a *minor* change. **VersionName** (a String, can be anything) defaults to 1.0 and is increased by 1 for every *major* change and 0.1 for every *minor* change. For example, an initial could be 1.0 which can be updated to 1.1 after a small change and 2.0 after a larger change. You will need to increase the VersionCode and change the VersionName of your application when you upload a new version to the Play Store.

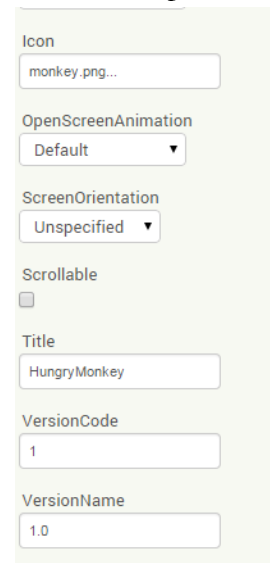


Fig 10.6 Screen – VersionCode and Name



## Review

- AI provides different ways to package and present your apps.



# Index

## A

About this Application .....	23
AboutScreen .....	23
AccelerometerSensor .....	13, 88
ActivityStarter .....	90
Address .....	89
AfterPicking.....	11, 12, 34
AfterPicture .....	72
AI coordinate system .....	30, 36
aiStarter .....	3
Altitude .....	89
Animation .....	71
Any component .....	21
Application software.....	2
Array .....	57
ASCII.....	67
Azimuth.....	90

## B

Back End (AI Blocks Editor) .....	3, 4, 14, 21
Ball .....	13
BarcodeScanner .....	13, 88
Behavior .....	39
Blocks .....	17
Boolean .....	35, 36
Built-in .....	20
Button .....	7, 11

## C

Camcorder .....	12
Camera.....	12
Canvas.....	13, 36
CheckBox .....	11, 32, 36
Clock .....	14
Colors .....	20
Comment .....	40, 43
Components .....	6, 39
Computers .....	1
Connectivity.....	90
Consumers .....	1
Control .....	20
Control Flow .....	41
Creators .....	1

## D

DatePicker .....	11
------------------	----

## E

Emulator .....	18
Event handler .....	38
Event-driven programming paradigm .....	38
External event.....	39

## F

For each .....	48
Front End (AI Designer).....	3, 4, 14

## G

Geocoding.....	89
Gmail account.....	7

## H

Hardware .....	1
Heading .....	71
Horizontal Arrangement.....	12

## I

Icon .....	23
If-then .....	41
Image .....	11
ImagePicker .....	12
ImageSprite .....	13
Infinite loop .....	53
Initialization event .....	39
Interval .....	71

## J

Join .....	27
------------	----

## L

Label .....	7, 11
-------------	-------

Latitude .....	89
List .....	20, 57
ListView .....	11, 32, 36
ListView .....	11
LocationSensor .....	14, 89
Logic .....	20
Longitude .....	89

## M

Math .....	20
Media .....	6
Method .....	40
MIT AI2 Companion App .....	3
Mobile Devices .....	1
Multiple screens .....	77
Mutator .....	27, 36

## N

Near Field Communications (NFC) .....	89
NearField .....	14
nested if .....	44
Notifier .....	11, 90, 91, 93

## O

Object .....	40
Object-Oriented Programming (OOP) .....	40
Operating system .....	1, 2, 4
Operator .....	40
OrientationSensor .....	14, 89

## P

Palette .....	6
Password TextBox .....	11
PasswordTextBox .....	36
Pitch .....	90
Pixels .....	71
Player .....	12
procedural (or functional) programming .....	40
Procedure .....	40
Procedures .....	20, 39
Properties .....	6
Property .....	40

## R

Roll .....	89
------------	----

## S

Screen .....	11, 36
Sensors .....	88
Share your app .....	97
Slider .....	11
Software .....	1, 2, 3, 4, 38
Sound .....	7, 12
SoundRecorder .....	13
SpeechRecognizer .....	13
Speed .....	71
Spinner .....	11
Sprite Z-layering .....	58
Statement .....	40
String comparisons .....	67
Systems software .....	1, 4

## T

Table Arrangement .....	12
TakePicture .....	72
Text .....	20
TextBox .....	11, 36
TextToSpeech .....	13
TimePicker .....	11
Timer event .....	39
TinyDB .....	72
Title .....	23

## U

URL .....	94
USB .....	3
User initiated event .....	39
User Interface (UI) .....	6

## V

Variable .....	39, 40
Variables .....	20
Vertical Arrangement .....	12
VideoPlayer .....	13
Viewer .....	6, 17

## W

WebView .....	11
While .....	51

## Y

YandexTranslate .....	13
-----------------------	----