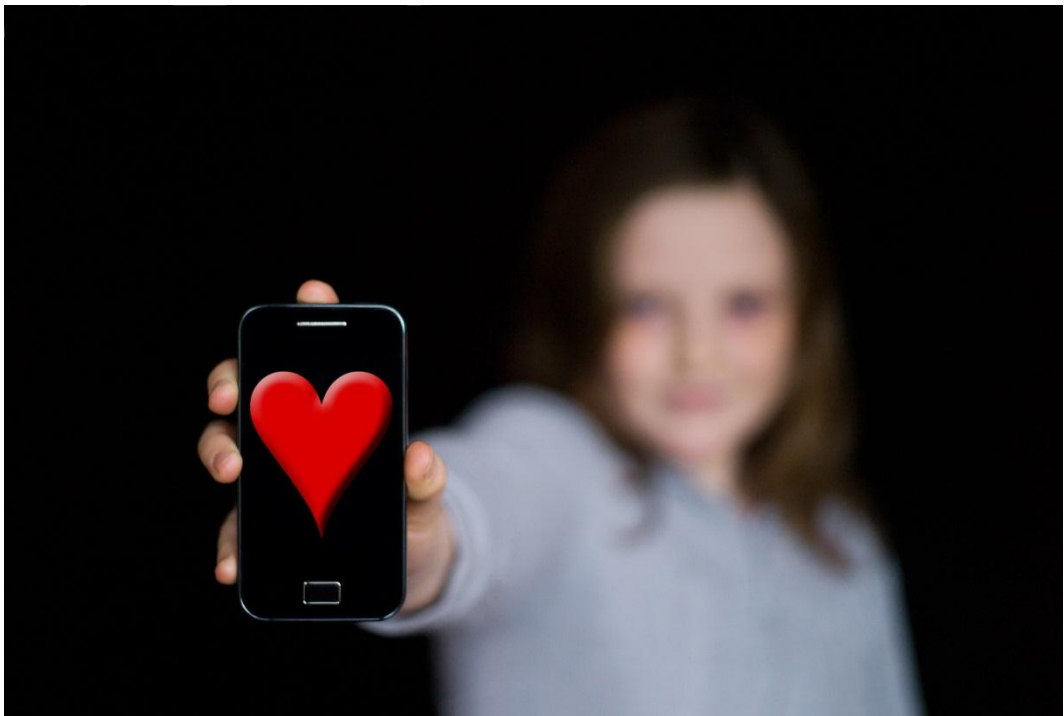


I ♥ My Smartphone



A Computing Science Course in Mobile App Development

by Jeremy Scott

LEARNER NOTES

Acknowledgements

This resource was partially funded by a grant from Education Scotland. We are also grateful for the help and support provided by the following contributors:

Bridge of Don Academy
Crieff High School
George Heriot's School
Johnstone High School
Kelso High School
CompEdNet, Scottish Forum for Computing Science Teachers
Computing At School
Professor Hal Abelson, MIT
Professor David Wolber, University of San Francisco
Scottish Informatics and Computer Science Alliance (SICSA)
Edinburgh Napier University School of Computing
Glasgow University School of Computing Science
Heriot-Watt University School of Mathematical and Computer Sciences
University of Edinburgh School of Informatics
Robert Gordon University School of Computing
University of Dundee School of Computing
University of Stirling Department of Computing Science and Mathematics
University of the West of Scotland School of Computing
ScotlandIS
Apps for Good
Brightsolid Online Innovation
JP Morgan
Microsoft Research
Oracle
O2
Sword Ciboodle

The contribution of the following individuals who served on the RSE/BCS Project Advisory Group is also gratefully acknowledged:

Professor Sally Brown (chair), Mr David Bethune, Mr Ian Birrell, Professor Alan Bundy, Mr Paddy Burns, Dr Quintin Cutts, Ms Kate Farrell, Mr William Hardie, Mr Simon Humphreys, Professor Greg Michaelson, Dr Bill Mitchell, Ms Polly Purvis, Ms Jane Richardson and Ms Caroline Stuart.

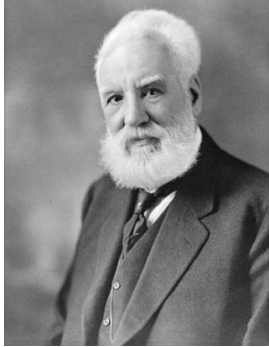
Some of the tutorials within this resource are based on existing material by Prof. David Wolber of the University of San Francisco and the App Inventor EDU site, reproduced and adapted under Creative Commons licence. The author thanks the individuals concerned for permission to use and adapt their materials.

Contents

A Brief History of the Telephone	1
Mr Watson, come here – I want to see you!	1
Going mobile	2
When mobile phones became smart phones	3
Convergence: Bringing it all together	4
Telephone...or computer?	5
Smartphone Software	7
Operating system	7
Apps	8
The mobile app industry	9
Mobile App Development	11
Lesson 1: Virtual Pet	12
Virtual Machines	14
Lesson 2: Finger Painting	16
The Importance of Design.....	16
Bugs.....	18
Variables.....	19
Lesson 3: Mole Masher Game	24
Procedures	25
Comments.....	26
Lesson 4: Times Table Helper	30
Validating Input.....	36
Lesson 5: Virtual Map Tour	38
Lists	40
Lesson 6: Heads I Win	42
Lesson 7: Wiff-Waff Game	46
Summary	51
Mobile App Project	53

A Brief History of the Telephone

Mr Watson, come here – I want to see you!



On March 10th 1876, Edinburgh-born inventor Alexander Graham Bell spoke into the first telephone: “Mr Watson, come here – I want to see you!”

His assistant, listening at the other end of the line in another room could hear Bell clearly. Bell had just invented the telephone and started a revolution in communication that would change the world.

The ability to speak directly to someone over a long distance quickly caught on and the land line phone quickly became the must-have accessory in more and more homes.

It remained difficult to call someone on the move, however. As early as 1930 it became possible to place a call to a passenger cruise ship via a radio link and but it was not until the outbreak of World War II that portable communications devices that we might recognise became available: first the backpack-based “Walkie-talkie” and then the hand-held “Handie-talkie” (opposite), both developed by Motorola.



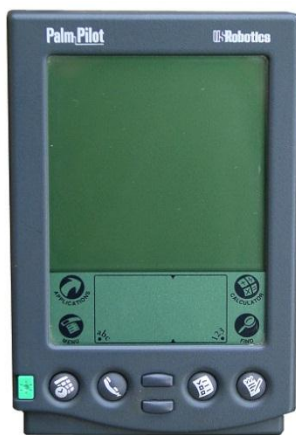
In the 1940s and 1950s, radio telephones in cars became available, but these were expensive, bulky and plagued with poor reception.

Going mobile

The first experimental hand-held mobile telephone was demonstrated in 1973 by Motorola and weighed around 1kg, but it was not until 1983 that the first devices went on sale.

These early devices were large and heavy, had a short battery life and were capable only of making telephone calls.

It was not until 1993 that manufacturers began to add extra features to mobile phones, such as a calendar, address book, clock, e-mail and simple games. The first of these was IBM's Simon handset.



Around the same time, hand held computers (often called PDAs – Personal Digital Assistants) began to appear that offered basic computing tasks in the palm of your hand.

These were commonly used with a stylus (like a pen) to tap a touch-sensitive screen. The most successful of these was the Palm range of handhelds (left). These devices couldn't make phone calls or send text messages – they were simple hand-held computers. They were, however, very popular and made Palm one of the world most successful technology companies of the late 1990s.



Investigate the products below. Complete the table by writing down what you think are the **two** most important features of each device.

Year	Product	Important features
1993	IBM Simon	
1996	Palm Pilot	
1998	Nokia Communicator	
2002	RIM Blackberry	
2007	Apple iPhone	
2008	HTC Dream	

When mobile phones became smart phones

These early smartphones were...well, not that smart. They could send and receive emails and perform a range of basic office functions for business users, but were not particularly user-friendly.

In 2007, Apple introduced the iPhone, which was aimed initially at consumers. The iPhone was a genuine “game-changer” and was the first mobile phone to feature a **multi-touch interface**. Until then, touch screens had been available on many devices, (such as the Palm Pilot) but could sense only one point of touch at a time.

The ability to sense more than one point of touch on a screen may seem like only a small improvement on single-point touch, but think of all the actions you can perform on a modern multi-touch device. It brought about the possibility of **gesture-based interfaces** where you can swipe, pinch and zoom with your fingers.

In just a few years, multi-touch interfaces have spread to many devices and have become steadily cheaper. It is difficult to foresee us ever going back to devices with lots of physical buttons to press.



An iPhone 3GS showing its large multi-touch screen.

Convergence: Bringing it all together

Modern smartphones bring together technologies such as GPS, Bluetooth and the accelerometer – and, of course, wireless Internet access.

This is an example of **convergence** – that is, integrating technologies which were once separate into a single device.



Investigate the following technologies. Beside each one, write down its **function** (what it does) and an **example** of a portable device (other than a smartphone) which uses it.

Microphone *Function:* _____

Example device : _____

Loudspeaker *Function:* _____

Example device : _____

Touch screen *Function:* _____

Example device : _____

Accelerometer *Function:* _____

Example device : _____

GPS sensor *Function:* _____

Example device : _____

Bluetooth *Function:* _____

Example device : _____

CCD *Function:* _____

Example device _____

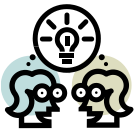


Discuss with your neighbour what you think will be the next big advances in smartphone technology. Using either a graphics package or pencil & paper, draw a labelled design for a smartphone 10 years from now.

Telephone...or computer?

Whilst we use the term “smartphone” to describe a modern multi-function mobile telephone, it is probably better to think of it as a pocket computer *which also happens to make phone calls*.

A computer is a machine which can follow a program (a list of instructions) to perform a task. As well as a **processing** device, it will usually have one or more **input**, **storage** and **output** devices.



Discuss with your neighbour what each of these could be within a smartphone.
Write down your answers.

Device	Smartphone example(s)
Input	
Processing	
Storage	
Output	

Did you know...? For many users, phoning is one of the tasks they perform the least on their smartphone.

To underline this, the 3 network claimed that by late 2011, 97% of all the traffic on its network was smartphone data¹. Voice calls and text messaging put together appear to have made up the remaining 3%.

Another report from network company Cisco stated that mobile data traffic in 2011 was eight times the traffic for the entire Internet in 2000. The same report predicted that by the end of 2012, there would be more mobile devices on the planet than people².

¹ Source: <http://blog.three.co.uk/2011/10/31/were-built-for-data/>

² Source: www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html

Smartphone Software

All of these technologies would be useless without **software** to bring them together. Software is the name for **computer programs** – list of **instructions** – which tell the computer hardware how to perform a task. On mobile devices, there are two main pieces of software:

- the **operating system** and
- **apps**

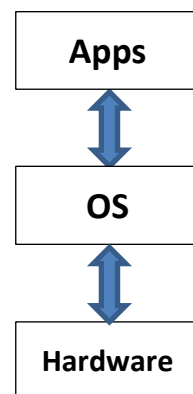
Operating system

An **operating system (OS)** is a set of programs that a computer runs all the time it is switched on.

An OS performs many tasks, but we can think of it as the program that gets the computer “up and running” and makes it work and look the way it does.

The OS sits between apps and the computer’s hardware, letting apps access the hardware. This is why it is possible to run software written for an OS such as Microsoft Windows on a wide variety of different computers.

Examples of operating systems on desktop computers include Microsoft Windows, Apple MacOS and Linux. At the time of writing, the most common operating system on mobile devices is Android from Google.



Write down the names of some **mobile device** operating systems. Beside each one, give the name of an actual device that it runs on.

If you or someone you know has a smartphone, write down what it is and the name of the operating system it runs.

Mobile OS

Actual Device

Android

Samsung Galaxy Ace mobile phone

Apps

Once your computer or smartphone has started up, you will want to do some tasks with it. Don't worry – no matter what you want to do, there's almost certainly an app for it!

The word "app" is short for **application**. An application is a computer program that enables the computer to do a useful job.



*Write down the name of three desktop applications.
Beside each one write down how much it costs.*

Now investigate a mobile app store such as Apple's App Store or Google Play. Write down the names of three mobile apps and their cost.

In most cases, you will see that a mobile device app has **fewer features** than a desktop application. This is usually reflected in the **price**, with many apps being free, or costing less than a pound.

The mobile app industry

Mobile app creation is still a very new industry, but is already worth billions of pounds per year. In December 2011, the number of monthly mobile app downloads exceeded 1 billion (1,000 million) for the first time, with 81 million of those in the UK alone.

Whilst many mobile apps are created by professional software development companies, there are also many “bedroom developers” – people who create mobile apps in their spare time, either as a hobby or a sideline to their main job. It is also an industry dominated by young developers.

In February 2012, it was reported that since the iPhone was introduced in 2007, the “app economy” had created an estimated 466,000 jobs in the USA alone³.

Maybe you will go on to make a career in this industry!

³ Source: *Where The Jobs Are: The App Economy*, Dr Michael Mandel, South Mountain Economics
<http://www.technet.org/wp-content/uploads/2012/02/TechNet-App-Economy-Jobs-Study.pdf>

Mobile App Development

The rest of this course will focus on how to write programs to create mobile apps.

You will be using **App Inventor**, originally created by Google and now taken over by MIT (Massachusetts Institute of Technology), one of the USA's leading universities.



App Inventor is a powerful software development package that lets you create apps for Android smartphones. We will be learning how to use App Inventor through a series of lessons.

At the end of each task, there will be some questions which will assess your understanding of what you have learned.

Lesson 1: Virtual Pet

This lesson will cover

- The App Inventor environment
 - Components
 - Properties
 - Code (blocks)
- Event-driven programming

Mobile features

- Touch interface
- Working with sound



Introduction

Your teacher will demonstrate a simple mobile app to you. This is the app that you are going to create – an on-screen virtual pet which you have to “care” for.



Task 1: Getting started with App Inventor

Watch screencast **VirtualPet1**. This will introduce you to App Inventor and the designer screen, where you assemble your app’s components.

Once you have done this, try creating a similar screen for your own VirtualPet app. Feel free to experiment with the different properties of each component!

Task 2: Creating code

At this point, all we have created is the app’s **interface**. Whilst this is an important stage in development, the app won’t actually do anything! We must now add some **instructions** (or **code**) to make the app work as intended.



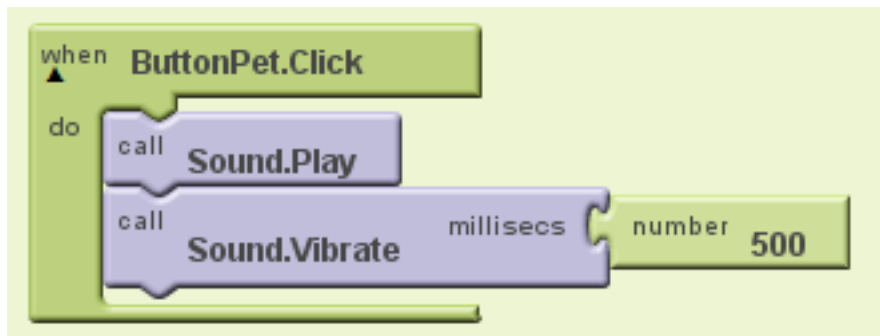
Watch screencast **VirtualPet2**, then try to create your own version of the app. If you get stuck, go back in the screencast or ask your partner.

There are **two** parts to an App Inventor app:

1. **Components** and their **properties** (or settings);
2. **Code** (or blocks) – the program **instructions**.
Most of the code is triggered by **events** – things that happen on the phone – such as a button click. Code can also be used to change the properties of the components.

Components need code (instructions) to perform a task.

Your code should look similar to the picture below:



Task 3: Testing your app

Using either the Android emulator or live testing using a mobile phone, try out your app.

Did it work as intended? If not, go back and check your code for any mistakes. If that doesn't work, check the properties of your app's components too.



Congratulations – you have just created your first mobile app!

Virtual Machines

The Android emulator is a recreation of a complete Android mobile phone on your computer.

It is obviously not a real phone – it can't make calls or receive texts, for example. However, App Inventor sees it as one, and sends the same commands to the emulator during testing as those sent to a phone.

The Computing term for this is a **virtual machine**.



Task 4: App Inventor reference sheet



Your teacher will issue a handout of the App Inventor environment. Fill this in and return it to your teacher.

Once your teacher has returned it to you, keep it handy for reference.

Extension Task 1

This virtual pet reacts to being touched, rather than being stroked. Your teacher will now demonstrate how to replace the **Button** component with a **Canvas** component, which can detect a **drag** event.

Now alter your app so that the cat meows only when it's being stroked.

Extension Task 2

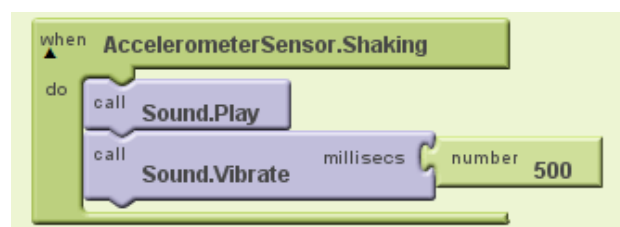
In the Designer, add an image sprite (**Animation**→**ImageSprite**) under the cat's chin. Make it big enough to detect a touch. Just leave it blank, with no image file.

In the Blocks Editor, use a **Sound.Vibrate** block from your sound component's drawer, make your cat purr (i.e. make the phone vibrate) when you stroke under its chin!



Extension 3 – Cool Feature

In the Designer, add an accelerometer component (**Sensors**→**AccelerometerSensor**). Then add the code shown opposite so that your virtual pet works when you shake the phone.





Did you understand?

It would be a simple task to add buttons that change the picture and sound for different animals. Such an app could be used to teach very young children how to recognise different animals and the sound they make, for example.

Run the app **VirtualPet3**, where this has been done. The code is shown opposite.



Unfortunately, there is a **bug** in the app – a problem that stops the app working as expected.

(Bugs are described in more detail in the next lesson)

```

when ButtonCat.Click
do
  set CanvasAnimal.BackgroundImage to text cat.png
  call Sound.Play

when ButtonDog.Click
do
  set CanvasAnimal.BackgroundImage to text dog.jpg
  call Sound.Play

when CanvasAnimal.Dragged
  startX name startX
  startY name startY
  prevX name prevX
  prevY name prevY
  currentX name currentX
  currentY name currentY
  draggedSprite name draggedSprite
do
  call Sound.Play

```

1.1 Try out the app and describe the problem.

1.2 What change would you have to make to the code to correct it?

1.3 Now make your change. Did it fix the problem? Did you have any problems doing this?

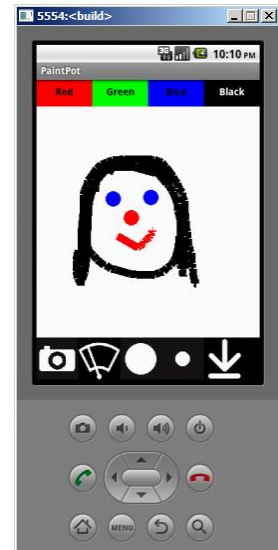
Lesson 2: Finger Painting

This lesson will cover

- The App Inventor environment
 - Components
 - Properties
 - Code (blocks)
- Event-driven programming
- Variables

Mobile features

- Touch interface
- Graphics
- Camera
- Accelerometer



Introduction

Your teacher will demonstrate a touch-driven finger painting app.



Task 1: Building the interface

Watch screencast **FingerPaint1** which covers building the app's interface.

The Importance of Design

Before we make anything – a house, a dress or a mobile app – we should start with a **design**. Because there are two important parts to a mobile app – the **interface** and the **code** – we design these separately.

- The easiest way to design the **interface** is by sketching it out on paper.
- The most common way of designing the **code** is to write out in English a list of steps it will have to perform. This is known as an **algorithm**.

Writing an algorithm is the key to successful programming. In fact, this is what programming is *really* about – solving problems – rather than entering commands on the computer.

All good programmers design algorithms before starting to code.

Task 2: Designing and creating the code

Algorithm & Code

when a colour button is clicked
set the canvas paint colour to
appropriate colour

when canvas is touched
draw a circle at that location

when wipe button is clicked
clear canvas

when small button is clicked
set line width to 5 pixels

when large button is clicked
set line width to 15 pixels

when canvas is dragged over
draw a line from the start point to
the end point of the drag

The image shows a series of Scratch code blocks for a drawing application. The blocks are organized into several groups:

- Color Selection:** Three 'when clicked' blocks for 'ButtonBlue', 'ButtonGreen', and 'ButtonRed'. Each block contains a 'do' block that sets 'DrawingCanvas.PaintColor' to the corresponding color (Blue, Green, or Red).
- Circle Drawing:** A 'when touched' block for 'DrawingCanvas'. It contains a 'do' block that calls 'DrawingCanvas.DrawCircle'. The 'x' and 'y' coordinates are taken from the 'touchedSprite' object, and the radius 'r' is set to the number 10.
- Wipe Button:** A 'when clicked' block for 'ButtonWipe' that calls 'DrawingCanvas.Clear'.
- Line Width Selection:** Two 'when clicked' blocks for 'ButtonSmall' and 'ButtonBig'. They set 'DrawingCanvas.LineWidth' to the numbers 5 and 15, respectively.
- Line Drawing:** A 'when dragged' block for 'DrawingCanvas'. It contains a 'do' block that calls 'DrawingCanvas.DrawLine'. The 'x1' and 'y1' coordinates are taken from the 'prevX' and 'prevY' properties of the 'draggedSprite' object. The 'x2' and 'y2' coordinates are taken from the 'currentX' and 'currentY' properties of the 'draggedSprite' object.



Watch screencast **FingerPaint2** to see how to build the code (shown above).



Bugs

A **bug** is an error which stops your code working as expected. There are **two** main types of bug which can occur in a program:

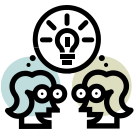
- **Syntax error**
This happens when the rules of the language have been broken, e.g. by misspelling a command. Syntax errors usually stop the code from running. Languages like App Inventor provide code in ready-written blocks, so you won't make many syntax errors.
- **Logic error**
This means your code runs, but doesn't do what you expect. Unfortunately, it's still possible to make logic errors in App Inventor!

Finding and fixing these errors in a program is known as **debugging**.

The code we have created so far has **two** bugs:



- The starting colour of the "paint" is black. Once you select another colour, there is no way to get back to black.
- The starting colour of the line is 1 pixel wide. Once you select the big or small brushes (5 or 15 pixels), there is no way to get back to 1 pixel.



Did you understand (part 1)?

2.1 Discuss with your partner what features your app would need to solve each of these problems. **Write your suggestions below:**

a) Paint colour: _____

b) Line width: _____



2.2 Discuss what you could do to reduce the chance of logic errors appearing in your apps. **Write your suggestions below.**



Task 3: Fixing the bugs

Your teacher will show you how to fix these bugs. Once you have seen this, make these changes to your FingerPaint app.

Did you know...? It is often said that the word “bug” dates back to 1947 when an early computer at Harvard University broke down because of a moth stuck in a switch! Whilst this **did** happen, the use of the word “bug” to mean an error or problem with a machine was used as far back as the 1800s.

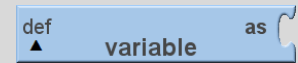


Variables

A **variable** is a space in a computer’s memory where we can hold information used by our program. It’s just like storing something in a box.

We should always give a variable a sensible **name** that indicates the kind of information that’s been stored there...just like putting a label on the box to tell us what’s inside.

To create (or define) a variable in App Inventor, we use the **def** built-in **Definition block** and then plug in a value to tell the computer what **type** of data it will contain such as text or number. Remember to give the variable a sensible name, too.



Once a variable is defined, the information stored inside it can be **changed** (or varied) – hence the word “variable”.

Extension 1: More flexible brush size

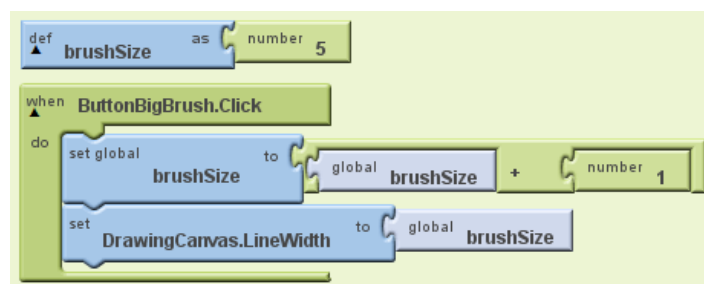
We have seen how we can use variables to store information such as the “brush” size. This makes our program code more **flexible**.

Now alter your app so that every time the user clicks on **ButtonBigBrush** or **ButtonSmallBrush**, the size of the brush is increased or decreased by 1 pixel.

Hint: Create a variable for the brush size (call it **brushSize**).

When **ButtonBigBrush** is pressed, our code should say “add one to brushSize”.

So...
$$\text{brushSize (new value)} = \text{brushSize (current value)} + 1$$



So, if brushSize was previously 5, it would now be equal to 6. Another click would make it 7, and so on. Once it has been set, we set the canvas line width to this value.

Once you have done this, create the code for **ButtonSmallBrush**.

Extension 2: Any colour you like

Add another horizontal row of colour buttons below the current ones.

Hint: by setting width to **Fill parent...**, you can also fit more colour buttons in a row – especially if you remove the **Text** property in the button.



Extension 3: Cool feature

Let’s add another feature: a **camera** button.

This will take the user will be taken to the camera app on the phone. After taking a photo, this will become the background of the drawing canvas. The user can then paint on the photo!

Hint: Add a camera button to the bottom row on the screen (a camera icon is provided with this lesson’s graphics). You will also need a **camera component (Media→Camera)**.



/...

Let's consider how we can make this happen and the code we'd use:

Algorithm

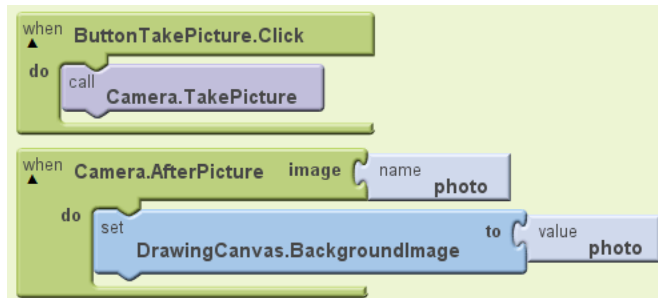
when camera button is clicked

take a picture

after picture is taken

set background image of canvas to the camera photo


Code



Extension 4: Another cool feature

Okay – one last feature: a **Save** button.

This will let users save their masterpieces to the image gallery on their phone!

Hint: add a save button to the bottom row on the screen (provided with this lesson's graphics – see down arrow icon opposite). You will also need a **TinyDB component** – this is used to store data permanently on the phone (**Basic**→**TinyDB**). 

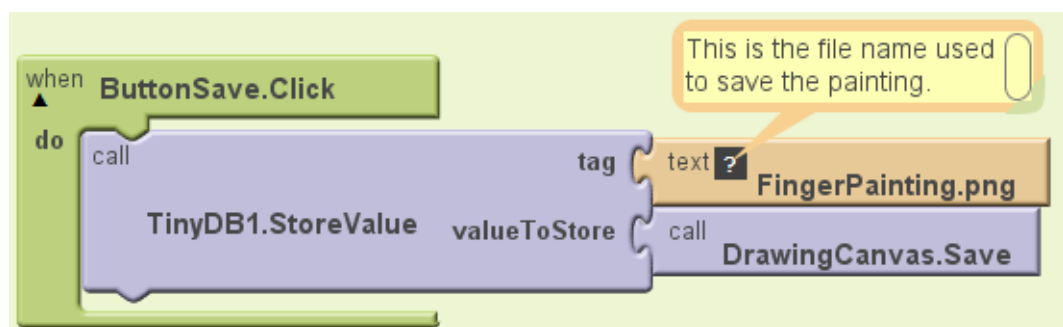


Note that this feature only works when the app is downloaded to the phone. It will not work under live testing or on an emulator.

Algorithm

when Save button is clicked

store drawing canvas image as file





Did you understand (part 2)?

In this lesson, we learned that variables can be used to store values in a program.

2.3 What **type** of variable – **text** or **number** – should be used to store the following values:

a) 23 _____

b) Alice _____

c) 3.14 _____

d) SG12 RDW _____

e) Fourteen _____

2.4 Using short variable names like **a**, **b** or **c** seems like it could save a programmer time and effort in typing. Why would this be a **bad** idea?

2.5 A variable name should always be as **meaningful** as possible – that is, the name should suggest the value that’s being stored. However, we shouldn’t make it longer than necessary.

Write down suitable variable names for the **names** and **scores** for two players in a game (**four** variables in all)

2.6 /...



2.6 Think of some non-Computing examples of “variables” and their possible values. An example is shown below:

Variable: **cutlery** Possible values: **knife; fork, spoon**

Variable: _____

Possible values: _____

Variable: _____

Possible values: _____

2.7 A user starts up a FingerPaint app and immediately clicks **ButtonBigBrush** (code shown below).

```

def brushSize as number 0
when ButtonBigBrush.Click
do
  set DrawingCanvas.LineWidth to global brushSize
  set global brushSize to global brushSize + number 1

```

However, when the user tries to paint, nothing appears on the canvas until they click **ButtonBigBrush** a **second** time.



Discuss with your partner why this happens and what change(s) should be made to the code to fix this bug.

Reason _____

Correct code _____

Lesson 3: Mole Masher Game

This lesson will cover

- Timers
- Variables
- Procedures

Mobile features

- Touch interface
- Working with graphics
- Using timers in games

Introduction

Your teacher will demonstrate a simple mobile version of a popular fairground game (commonly known as Whack-a-Mole). In this game a mole appears and you have to tap it quickly before it disappears again. This is the app that you are about to create.

No moles will be harmed in the making of this app! However, you can use a graphic of a target (file: Target.png) if you prefer – or even a draw a character of your own!



Task 1: Creating the interface

Watch screencast **MoleMasher1**. This will take you through creating the interface – the screen design and components – needed for the game.

Task 2: Designing the solution

Let's consider the main steps we need to code in our game. There are **two** main stages:

- touching the mole
- moving the mole

We'll now design our code by creating an **algorithm** for each stage.

Algorithm

To Update Score Display (procedure)

set the text of the score label to "Score: " + the score

To Move Mole (procedure)

set the mole's X co-ordinate to a random place along the canvas

set the mole's Y co-ordinate to a random place down the canvas

When the start button is clicked

set the score to zero

call Update Score Display procedure

When the mole is touched

increase the score

call Update Score Display procedure

make the phone vibrate

call Move Mole procedure

Every second (1000 millisecs) during the game

call Move Mole procedure



Watch screencast **MoleMasher2** to see how to build the code from the algorithm above.



Procedures

In this lesson, we saw how lines of code can be grouped together into a **procedure**. Creating a procedure is like **creating a new command** in your programming language.

Procedures let us:

- break down a problem into smaller problems and solve each of those separately. We can then concentrate on just one small "sub-program" at a time.
- create a single piece of code that we can use (or **call**) as often as we need to within a program. This saves us "reinventing the wheel" by entering the same code lots of times.

As a general rule, whenever you have a clear "sub-task" in your program, you should create a procedure to do this. It will make your life easier!

Extension 1

Adapt your program to display the number of **misses** as well as the number of hits.

Extension 2

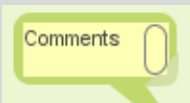
Display a “GAME OVER” sprite on the canvas when the misses reach a certain number.

Hint: Create a GameOver procedure for this.



Comments

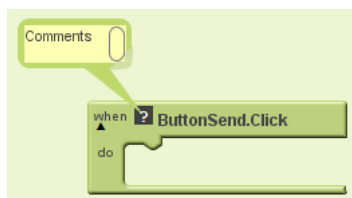
You may have noticed that some code is shown with small comments beside it in speech bubbles.




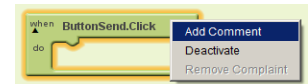
Comments are used to **explain what code is doing**. This is useful if you’re working as part of a team, so that other programmers can understand your code – or even for yourself, when you try to update your app this time next year!

All good programmers use comments to explain key stages in a program.

In App Inventor, right-click on a code block to add a comment.



Once you have done this, click inside the speech bubble to add a comment. You can show/hide comments by clicking the black question mark icon  on a code block which has a comment.



Remember: comments are there to help you and can save hours of frustration when you’re trying to understand another programmer’s (or even you own) code!

Extension 3

Go back to your code and add comments to it.

Note that you would normally add comments as you create your code, not afterwards.



Did you understand?

3.1 How could you make the game more difficult for users?



3.2 Discuss the following examples from real life. What “procedures” could they be broken down into?

a) Getting ready for school _____

b) Making breakfast _____



3.3 Discuss with your partner some examples of sub-tasks within a simple “space invader”-type game that could be coded as procedures.

Write down the “sub tasks” in the space below.



3.4 A user scores 10 hits and 5 misses in a MoleMasher app, then clicks a Reset button (code shown below).

```
when ButtonReset.Click
do
  set LabelHitsNumber.Text to global hits
  set LabelMissesNumber.Text to global misses
  set global misses to number 0
  set global hits to number 0
```

Write down the values displayed in **LabelHitsNumber** and **LabelMissesNumber** after the Reset button is clicked.

LabelHitsNumber: _____

LabelMissesNumber: _____

Write down any changes you would make to the code for the Reset button below:

when ButtonReset.Click do

Lesson 4: Times Table Helper

This lesson will cover

- Handling user input
- Variables
- Fixed loops
- Validating input
- Working with text

Mobile features

- Touch interface
- Using the keypad

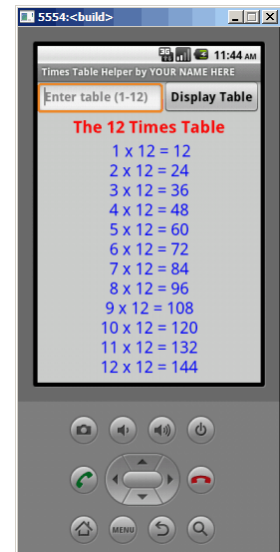
Introduction

Know your times tables? All of them...? How's your 47 times table?!

In this lesson, we're going to create a handy times table reminder that goes way beyond the 1 to 12 times tables.

Task 1: Creating the interface

Create the interface for the app, as shown overleaf (no screencast this time – try it yourself):



Horizontal arrangement containing	
Text Box + non-default properties...	
Name	TextBoxTableNumber
Hint	Enter table (1-12)
Numbers only	<i>[Tick]</i>
Font	Bold, 18, sans serif, black, left aligned
Size	Width: Fill parent Height: Automatic
Button + non-default properties...	
Name	ButtonCreateTable
Text	Display Table
Font	Bold, 18, sans serif, black, centre aligned
Size	Width: Fill parent Height: Automatic

Label + non-default properties...	
Name	LabelTableTitle
Text	<i>[Blank]</i>
Font	Bold, 24, sans serif, red, centre aligned
Size	Width: Fill parent Height: Automatic

Label + non-default properties...	
Name	LabelTimesTable
Text	<i>[Blank]</i>
Font	24, sans serif, blue, centre aligned
Size	Width: Fill parent Height: Automatic

Make sure the **Screen1** component is set to **Scrollable**.

Task 2: Designing the solution

Let's consider the main steps we need to code in our game. There are **three** main stages:

- Getting the times table number from the user via the text box
- Creating the times table header e.g. **The 12 Times Table**
- Creating the times table itself

Algorithm**When button "Display table" clicked**

set a variable <table> to the number entered in the text box
 call **Create Table Header** procedure
 call **Create Table** procedure
 clear the text box

To Create Table Header (procedure)

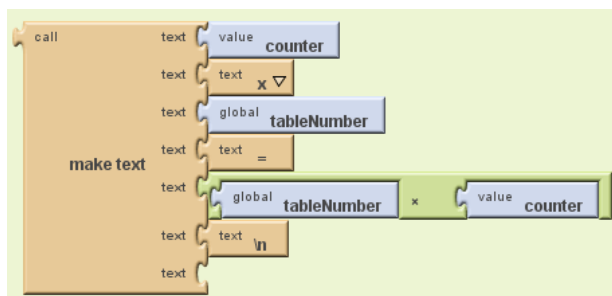
set the table header label's text to "The" <table> "Times Table"

To Create Table (procedure)

repeat 12 times using a counter
 create a new line in the table
 add this line on to the previous table text

To create a new line in the table we create the following block of text:

loopCounter	x	tableNumber	=	loopCounter x tableNumber
variable	text	variable	text	math
<i>value</i>		<i>value</i>		<i>result of calculation</i>



(text block `\n` = take a new line)

So, if the user enters 5 for the table number, it will create the text:

$$\begin{array}{l} 1 \times 5 = 5 \\ 2 \times 5 = 10 \\ 3 \times 5 = 15 \end{array}$$

...and so on.



Now watch screencast **TimesTable**. This will take you through creating the code that implements the algorithm above.

Extension 1: Weight converter

Design and write an app to create a conversion table of kilograms to pounds (1kg = 2.2 lb). The table should go from :

a) 1 to 10

b) 5 to 100 in steps of 5 (use the STEP feature in the loop)

e.g.

Kilogrammes to Pounds conversion

1kg = 2.2lb

2kg = 4.4lb

3kg = 6.6lb

...etc.

NB There is no input from the user in this app. It should just work when you click a “Display conversion table” button

Extension 2: Currency converter

Design and write an app that lets the user enter the current exchange for US Dollars and produces a table to US Dollars to Pounds.

Choose suitable start and end points for the amounts, as well as an interval (step) for your table. Use a search engine to find out the current exchange rate when testing your app.



Did you understand (part 1)?

4.1 From a programming point of view, why is it a good idea to let the user enter the table via an input box and store this in a variable?

There is a bug in the Times Table app.

After creating the first times table, it keeps adding new tables on to the end of the previous one every time the display table button is clicked, instead of replacing the current times table.



If you haven't already noticed this, try out the app again. You may have to scroll down on the phone/emulator to see this, so make sure the **Scrollable** property is ticked in your **Screen1** component.



4.2 Discuss why you think this happens. Write your reason below.



4.3 Discuss what change you would need to make to the code to prevent this from happening. Describe it below.

4.4 After discussing your answer to 4.3 (above) with your teacher, make the change to your code.

Did it fix the problem? _____

If not, what mistake did you make? _____



Did you understand (part 2)?

There is another bug in the times table app.

Try clicking the Display Table button without entering a number. Your app will display an error message then quit. This is known as a program **crash**.



Don't crash and burn!

An app which crashes will attract very few users!



4.5 Discuss why you think the app crashed. Write your reason below.



4.6 Discuss what could be done to prevent the app from crashing. Write down your suggestion below.

Now read on to see how to do this...



Validating Input

Whenever we get input in a program we should always check that it is **valid** – allowable or reasonable – before we process it.

If an input is **invalid**, we should:

- tell the user they have entered an invalid value
- tell them what the valid values are
- ask them to re-enter their input

The program should not progress until the user enters a valid value.

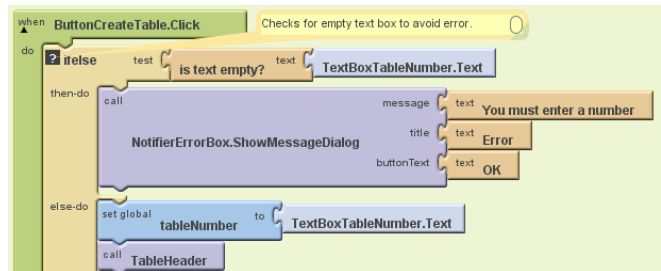


Now let's amend the app so that if the text box is empty, the user receives an error message. Our app will only create the times table if a number is entered.

Algorithm

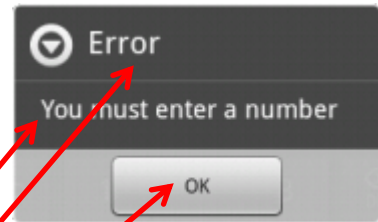
```

if text box is empty
    display error message
else
    display table
    
```



To display the error message, we will use a **Notifier** component (**Other Stuff**→**Notifier**), so add one to your **component screen**. The Notifier component is used to create a box on the screen with a message.

The code to make the Notifier is shown below – feel free to alter the text.



A Notifier is useful because it stops the user performing any other function in the app until they have dealt with the notification. This way, we can be sure the user has seen and (hopefully) read the message.

On a desktop computer, a notification is often called a **dialogue box**.



Did you understand (part 3)?

4.7 Write down a **range of valid values** for the following inputs:

a) someone's age _____

b) the number of days in a month _____

4.8 Write down an **invalid** value for a number that an app will divide another number with.



4.9 In this app, we saw the use of an **if...else** statement to tell whether or not the user had entered a value into the text box.

Now consider an app which decides whether students' test scores resulted in a pass or a fail (assume a pass mark of 50).

Which of the following **if...else** statements would produce the correct results?

Write the letters of the correct statements below the table.

A	IF testScore = 50 student has passed ELSE student has failed	D	IF testScore < 50 student has failed ELSE student has passed
B	IF testScore <= 50 student has failed ELSE student has passed	E	IF testScore ≠ 50 student has passed ELSE student has failed
C	IF testScore >= 50 student has passed ELSE student has failed	F	IF testScore > 50 student has passed ELSE student has failed

Correct results (letters) _____

Lesson 5: Virtual Map Tour

This lesson will cover

- Variables
- Lists

Mobile features

- Touch interface
- Working with lists
- Linking to external services

Introduction

In this example, we're going to create a guided tour that links in to Google Maps.

When the user clicks the "Explore Edinburgh" button, they will be presented with a list of popular sightseeing destinations within the city, each of which will bring up that location in the phone's Maps app. The user will then tap the phone's Back button to return to the Tour app.

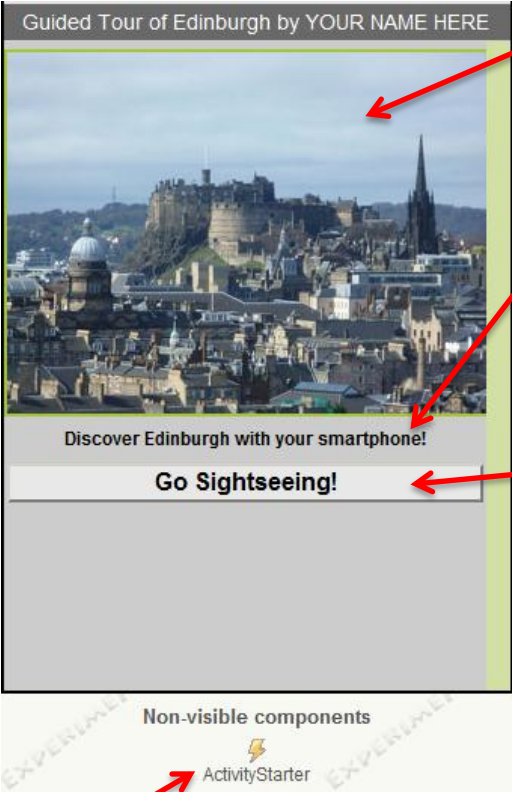
Task 1: Creating the interface

Create the interface shown overleaf:



/...

Task 1: Creating the interface



The screenshot shows the visual design of the app. Red arrows point from the interface elements to their respective property tables:

- The image of Edinburgh is linked to the **Image + non-default properties...** table.
- The text 'Discover Edinburgh with your smartphone!' is linked to the **Label + non-default properties...** table.
- The 'Go Sightseeing!' button is linked to the **ListPicker + non-default properties...** table.
- The 'ActivityStarter' component in the 'Non-visible components' section is linked to the **ActivityStarter + non-default properties...** table.

ActivityStarter + non-default properties...	
Action	android.intent.action.VIEW
ActivityClass	com.google.android.maps.MapActivity
ActivityPackage	com.google.android.apps.maps

This app makes use of the **ActivityStarter** component (**Other stuff**→**ActivityStarter**) which enables one app to start up other apps. In this case, our app will start up the Maps app on the phone.

Note that you must enter the properties for the ActivityStarter exactly as shown (including upper/lower case letters).

Task 2: Designing the solution

Let's consider the main steps we need to code our Virtual Map Tour. The app will let users pick a location from a list and display that location in the Maps app.

There are **two** main stages:

- setting up the lists
(one for place names and one for corresponding map references)
- launch Maps with location chosen from the lists

Algorithm

Set up the lists

- create variable containing a list of place names
- create variable containing a list of map references

Display the map

- note the position of the chosen location in the place names list
- open maps with the URI in the corresponding position in the locations list



Watch screencast **MapTour** to learn how to create the code needed for the app.

Extension 1

Alter your app to create a similar map tour for your own local area. Remember – you can use Google’s Street View to get up close!



Lists

Lists are ideal for storing items of data that are related, rather than having separate variables for each item. For example, we could store the name of everyone in a class in a single list instead of having separate variables for Student1, Student2, Student3, etc.

In this example, we worked with **two** lists:

- one that held the **place names** the user sees in the list picker
- one that held the **corresponding Google Maps locations** (URIs)

By keeping the two pieces of information in the corresponding position in each list, we could tie them together – that is place name #1 corresponded to map location #1, etc.

Position	Place names list		Google Maps locations list
1	Edinburgh Castle	→	http://maps.google.com/maps?q=Edinburgh+Castle+Grounds,+Castle+Terrace,+Edinburgh...
2	Museum of Scotland	→	http://maps.google.co.uk/maps?q=museum+of+Scotland&ll=55.94711,.191306&spn=0.0...
3	Scott Monument	→	http://maps.google.com/maps?q=Edinburgh+Castle+Grounds,+Castle+Terrace,+Edinburgh...

This is a common way of tying two or more pieces of information together in Computing Science.



Extension 2: Cool feature

Let's add **location awareness** to your map tour.

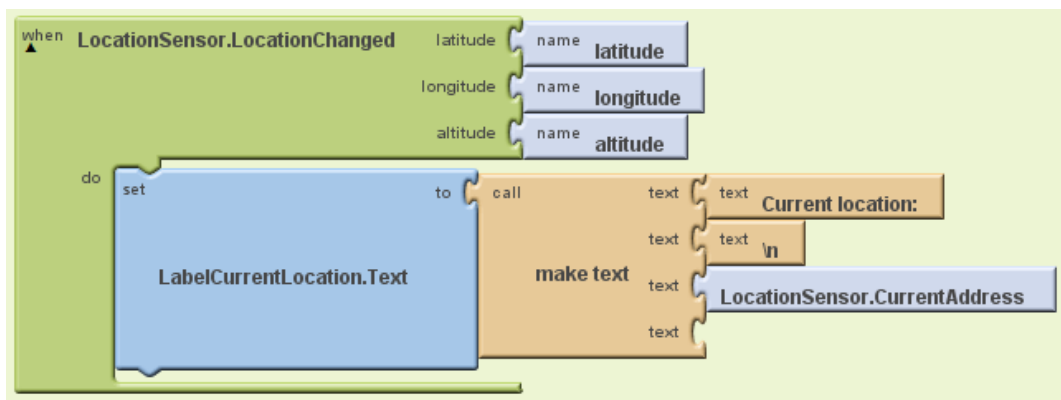
For this we will need a **LocationSensor** component (**Sensors**→**LocationSensor**).



Set the following properties:

LocationSensor + properties...	
Enabled	[Tick]
ProviderLocked	[Tick]
ProviderName	gps

Add a label (call it **LabelCurrentLocation**) below the ListPicker component on your screen, then add the following code in the blocks editor:



Your app should now provide a live readout of your current location!

Lesson 6: Heads I Win

This lesson will cover

- Variables
- Conditional loops
- Working with text

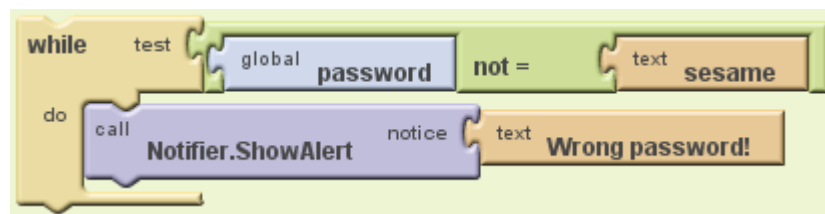
Mobile features

- Touch interface

Introduction

In Lesson 4: Times Table Helper, we used a FOR loop to repeat a piece of code a fixed number of times. For this reason, a FOR loop is sometimes called a **fixed loop**.

It is also possible to repeat code **while a condition is true**, ending only when something has happened e.g. the user has entered the correct password. This is called a **conditional loop**.



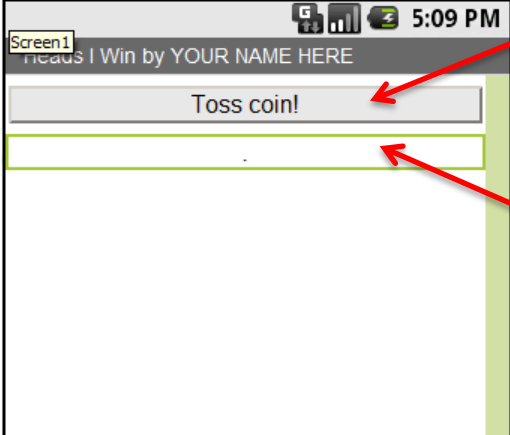
A conditional loop is used when we don't know in advance how often a process will have to be repeated.

In this lesson, we are going to create an app which simulates the toss of a coin and displays the output. The simulation will continue until Heads appears six times.



Task 1: Creating the interface

Create the interface shown below:



Button + non-default properties...	
Name	ButtonTossCoin
Text	Toss coin!
Font	Bold, 18, sans serif, centred
Size	Width: Fill parent; Height: Automatic
Label + non-default properties...	
Name	LabelOutput
Text	[Blank]
Font	18, sans serif, centred
Size	Width: Fill parent; Height: Automatic

Make sure your screen component is set to **Scrollable**.

Task 2: Designing the code

Let's consider the main steps we need to code in our app.

We will have **two** variables: one for the **side** (0 or 1, representing Heads or Tails) and one to **count the number of heads**. The app will continue tossing the coin until it has counted a total of six heads, then display the total number of tosses.

Algorithm

To Initialise (procedure)

```

set side to 0
set headsCounter to 0
set output label to <empty text>

```

when Toss Coin button is clicked

```

call Initialise procedure
while headsCounter is less than 6
    set side to a random number between 1 and 2
    if side = 1
        add 1 to headsCounter
        add "Heads" to the output label and take a new line
    else
        add "Tails" to the output label and take a new line

```



No screencast this time – try coding this one yourself!



Extension 1

Discuss with your partner how you would change this app so that it **displays the number of times the coin was tossed**. Write down your ideas below.

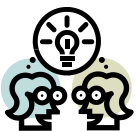
Discuss this with your teacher, then make the change to your app.



Extension 2

Discuss with your partner how you would change this app so that it continues until 6 heads **in a row** are produced. Write down your ideas below.

Discuss this with your teacher, then make the change to your app.



Extension 3

Discuss with your partner how you would change this app so that it continues until **either 6 heads in a row or 6 tails in a row** are produced. Write down your ideas below.

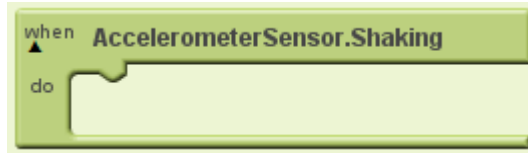
Discuss this with your teacher, then make the change to your app.



Extension 4: Cool feature

Let's add a great feature to this app – shake the phone to toss the coin!

Add an accelerometer component to your app (**Sensors**→**AccelerometerSensor**) and drag out a **when AccelerometerSensor.Shaking** block in the blocks editor.



However, instead of copying and pasting your code all over again from the **ButtonTossCoin.Click** block, can you think of a better way to do it?

Discuss this with your partner write down your idea below.

Now discuss this with your teacher, then make the change to your app.

NB All that you need to do is give the phone a gentle flick. If you shake it too much, the app will detect lots of Accelerometer shaking events and may crash!

Lesson 7: Wiff-Waff Game

This lesson will cover

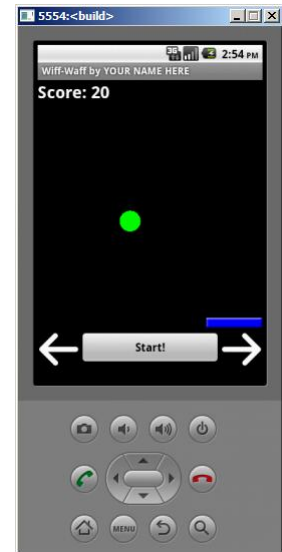
- Variables
- Procedures
- Collision detection

Mobile features

- Working with graphics
- Using timers in games

Plus, as an extension

- Motion detection and control



Introduction

Your teacher will demonstrate a classic bat & ball game. This is the app that you are about to create.

In this game, the bat will move continuously and the user will change its direction by clicking the arrow buttons at the bottom of the screen. This makes the game quite challenging, but more fun to play!

Did you know...? *Wiff-Waff* is the original name for table tennis and was a popular “parlour game” in Victorian times. After dinner, wealthy Victorians would use boxes to hit a golf ball back and forth across the dining table! The “net” was made by placing books on their ends.

Task 1: Creating the interface

Create the interface for this app as shown over leaf.

Hint: Set the Screen component's background colour to Black **at the very end**. It is easier to work with a white background as you assemble the components.

The screenshot shows the Wiff-Waff game interface. At the top, there's a status bar with signal strength, battery, and time (5:09 PM). Below it, a header bar says "Wiff-Waff by YOUR NAME HERE". The main area is black with a "Score:" label at the top left. A red ball is in the center, and a bat is at the bottom. A "Start!" button is in the middle. At the bottom, there are left and right arrow buttons. A "Non-visible components" panel at the bottom shows "SoundBeep" and "SoundLose" icons. Red arrows connect these elements to their respective property tables on the right.

Label + properties...	
Name	LabelScore
Text	[Blank]
Font	18, Bold, white text
Size	Width: Fill parent Height: Automatic

Canvas + properties...	
Name	CanvasGameArea
BkgdColor	None
Size	Width: Fill parent Height: 315 pixels

Ball + properties...	
Name	Ball
Interval	10
PaintColor	Red
Radius	15
Speed	7

ImageSprite (NOT Image)	
Name	ImageSpriteBat
Interval	10
Picture	Bat.png
Speed	7

HorizontalArrangement with...	
Button + properties...	
Name	ButtonLeft
Image	ArrowLeft.gif
Text	[Blank]
Button + properties...	
Name	ButtonStart
Font	Bold, 16
Text	Start!
Size	Width: Fill parent Height: Automatic
Button + properties...	
Name	ButtonRight
Image	ArrowRight.gif
Text	[Blank]

Sound + properties...	
Name	SoundLose
Source	BeepLose.wav
Sound + properties...	
Name	SoundBeep
Source	Beep.wav

Task 2: Creating the code

Let's consider the main steps we need to code in our game. There are **four** main stages:

- set up the game
- move the bat
- ball collides with bat
- ball reaches an edge

Let's design these stages by creating an algorithm for each one.

Algorithm

to Set Up Game (procedure)

set the score to 0
 set the ball enabled to true (lets it move)
 set the ball colour to green
 set the ball's coordinates to 150, 150
 set the ball's heading to a random number between 45 and 135 degrees
 disable the start button

when Start button is clicked

call Set Up Game procedure

when left button is clicked

set bat heading to 180 (left)

when right button is clicked

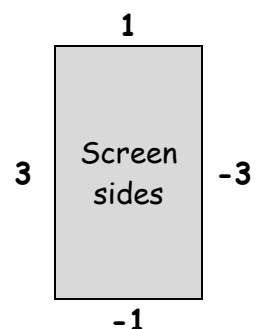
set bat heading to 0 (right)

when ball collides with bat (use `Ball.CollidedWith` block)

play beep sound
 set ball heading to ball heading - random number (say, 170 to 190)
 call Increase Score procedure

when ball reaches an edge (use `Ball.EdgeReached` block)

if edge = -1 (bottom of screen)
 call Lose Game procedure
 else
 bounce ball off edge
 play beep sound



Contd/...

to Increase Score (procedure)

add 10 to score
set Score label text to "Score: " + the score

to Lose Game (procedure)

play lose sound
play vibration for 500 millisecs
set ball paint colour to red
disable the ball (stop it moving)



No screencast this time – try coding this one yourself!

Did you understand (part 1)?



7.1 a) The algorithm above has a bug which shows itself when you lose. What is it?



b) Describe what you would have to do to your algorithm to fix it?
Hint: the change should be made in the **LoseGame** procedure.



c) Besides allowing us to concentrate on one sub-task at a time, write down **one** other benefit of splitting code up into procedures.
Hint: there is a clue in the answer to part b) above.



7.2 Why do we test for a collision at the bottom of the screen in the **if...else**, rather than the other sides?

Extension 1: Skill levels


Add buttons which allow the user to select levels: Easy, Medium or Difficult. Each of these buttons should adjust the speed of the ball to suit.



Extension 2: Cool feature

Let's add motion control to our game, so that tilting the phone will move the bat.

Note that this feature will only work on a phone.

You will need to add an **OrientationSensor** component to your app  (**Sensors**→**OrientationSensor**). In this app, we will use the sensor's **roll** value, which detects left and right tilt on the phone.

Algorithm

Moving bat (using Orientation Sensor)

```
if orientation sensor roll > 0 then (phone is tilted to the right)
    set the bat heading to the right (0)
else (phone is tilted to the left)
    set the bat heading to the left (180)
```



Did you understand (part 2)?

7.3 In what direction will the bat move if the phone's tilt is zero (completely level)?

Why?



Summary

Computing Science concepts

You have also learned about some important ideas within Computing Science:

- Computer software
 - Operating system
 - Apps
- Event-driven programming
- Algorithms
- Variables
- Procedures
- Input validation
- Virtual machines

Programming structures/commands

In this course, you have used the following programming structures:

- The App Inventor environment
 - Components, properties and code
- Decision-making
- Timers
- Variables
- Procedures
- Handling user input
- Variables
- Loops
 - Fixed (For)
 - Conditional (While)
- Validating input
- Working with text

There are, of course, many more, but you now have the necessary tools to go on to the next stage

Mobile device features

You have also learned about or how to access the following features of a smartphone:

- Touch screen interface
- Text
- Graphics
- Sound
- Animation
- Notification boxes
- Motion control
- Accelerometer
- GPS/location awareness

You now have all the skills you need to create some really amazing mobile apps.

So what are you waiting for?

Mobile App Project

Working in a pair or group, you are now going to **develop a mobile app of your own!**

You may have some ideas already, but software is normally designed by going through a series of stages:

1. Analyse
2. Design
3. Implement
4. Test
5. Document
6. Evaluate
7. Maintain



Or... **A** Dance In The Dark Every Midnight!

Let's consider what each of these stages means within mobile app development.

Analyse

Identify a problem or need your app is going to address. Think about its possible users. It's often best to choose something that interests you or you care about.

Design

Design your app in two stages:

1. **Make a sketch of the interface**
This is sometimes known as a **wireframe** and is the best way to get a clear idea of how your app will work.
2. **Design your code by creating algorithms**
DO NOT just start to try coding without designing it first! Remember the ancient programmer's proverb:

"Hours of coding can save minutes of design"

Implement

Create your app's screen and components, then create the code that corresponds to your algorithms.

Remember to **comment your code** so that it makes sense to other developers – and you, when you come back to fix bugs or add new features next year!

Test

- Test your app to make sure it works.
- Give it to other users and note their comments.
- Fix any bugs that are discovered.

Document

A desktop application comes with **documentation** – instructions on how to install and use it.

However, a mobile app should need little or no documentation. Often the only documentation is some simple instructions, either on the app itself, or on its page on an app store. You must therefore make your app as **intuitive** (obvious how to use) as possible.

An important part of documentation is ensuring that **comments** are included with code, although this should be done during coding.

Evaluate

When you've finished the task, make an honest assessment of how you did. Some questions that you might ask include:

- Did the app turn out as planned?
- What mistakes did you make on the way?
- If you were to start again from the beginning, what would you do differently?
- Are there any features that you think would make it better?

This is a vital stage in development because it is only by answering these questions that developers can improve their skills – and future apps.

Maintain

Maintenance is the process of updating your app. There are **three** main types of maintenance:

1. **Fixing bugs** that weren't discovered during testing
2. Adding **new features**
3. **Adapting your app** to work on new devices
A good example of this is creating a tablet version of an app originally written for smartphones.

Now let's go through these stages to create your own mobile app!

Analyse



Working in pairs or small groups, **brainstorm three ideas for your app.**

As you do so, think about each app's **possible users** and the **need it's going to fulfil**. Think of how it might link in with other subject areas you're studying.

1. _____

2. _____

3. _____



Now discuss your ideas with your teacher.

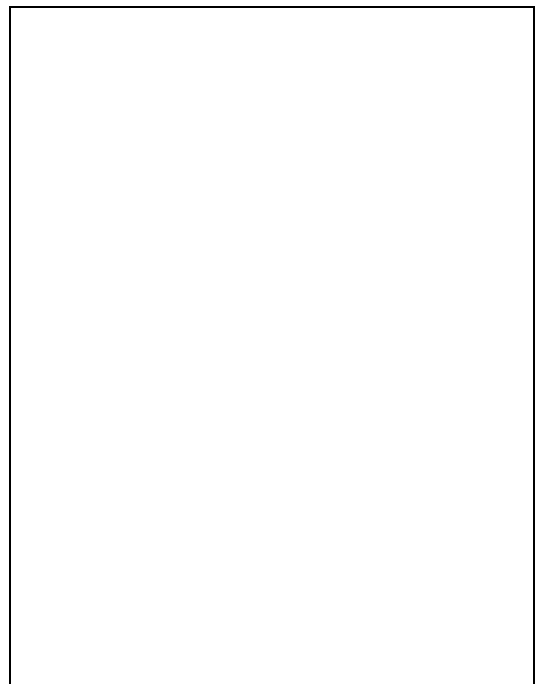
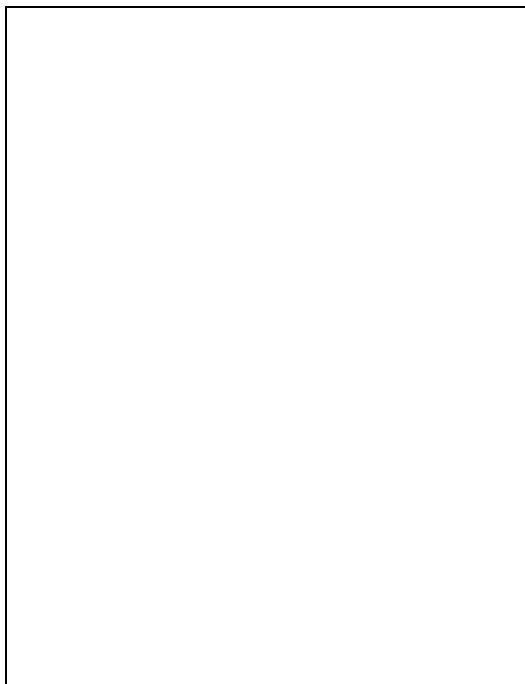
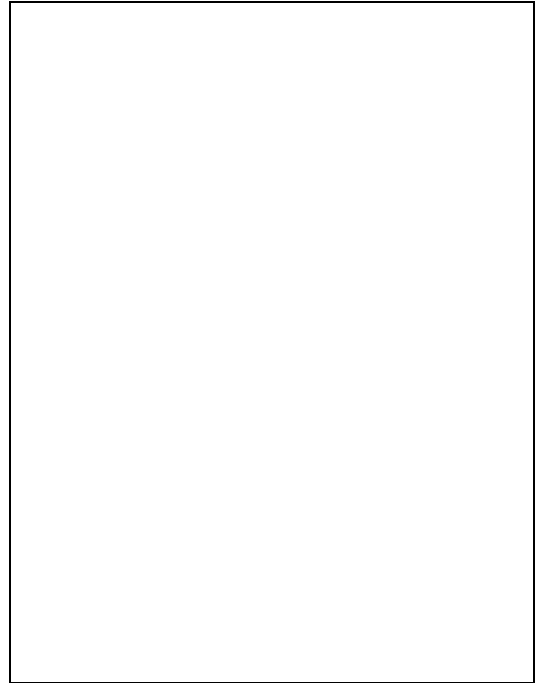
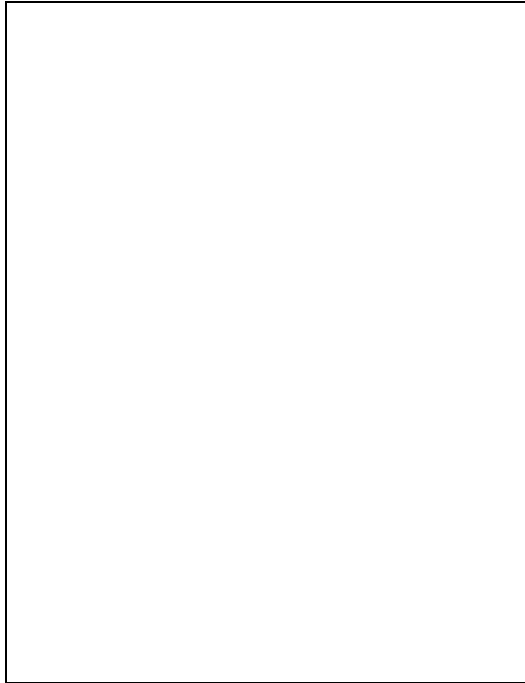
Once you have agreed on the app you are going to develop, write down a fuller description of what it will do below. Include any mobile features it will use.

Design (Interface)

Make a sketch of your app's interface.

Your sketch should be labelled to show what each component does.

Four screen layouts are provided to allow you to experiment.



Design (Code)

Design the algorithms for your code:

- Think about the steps **each component** from your screen design will have to perform. Write them in English.
- Use **procedures** where appropriate.
Remember, whenever you have a clear “sub-task” in your app, you should create a procedure to do this. This is especially true if you will use it more than once!

Implement

Now create your app!

- **Create the components in the designer**
Remember to give them sensible names.
- **Then create the code in the blocks editor**
Remember to include comments in your code and make sure you have your algorithm in front of you!

Test

Test your app to make sure it works.

Let your classmates test it too and note their comments below:

Describe bugs that were found (by you or by testers) and how you fixed them:

Bug: _____

Solution: _____

Bug: _____

Solution: _____

Bug: _____

Solution: _____

Document

Let's imagine that you're going to sell your app on an app store.

Write down below a brief description of:

- your app's **main features** and
- **how to use them.**

Remember – you're trying to get people to buy your app!

Evaluate

How did the app turn out **compared to how you originally planned it**?

What **mistakes** did you make on the way?

If you were to start again from the beginning, what would you **do differently**?

What **additional features** would make your app better?

Maintain

Now imagine that you have to adapt your app to make it work on a **tablet computer**.

What changes would you make?



Congratulations

You have now completed this Computing Science course in mobile app development!

Remember that you can use App Inventor at home, so there's no need for this to be the end of your time as a mobile app developer.



<http://appinventor.mit.edu>