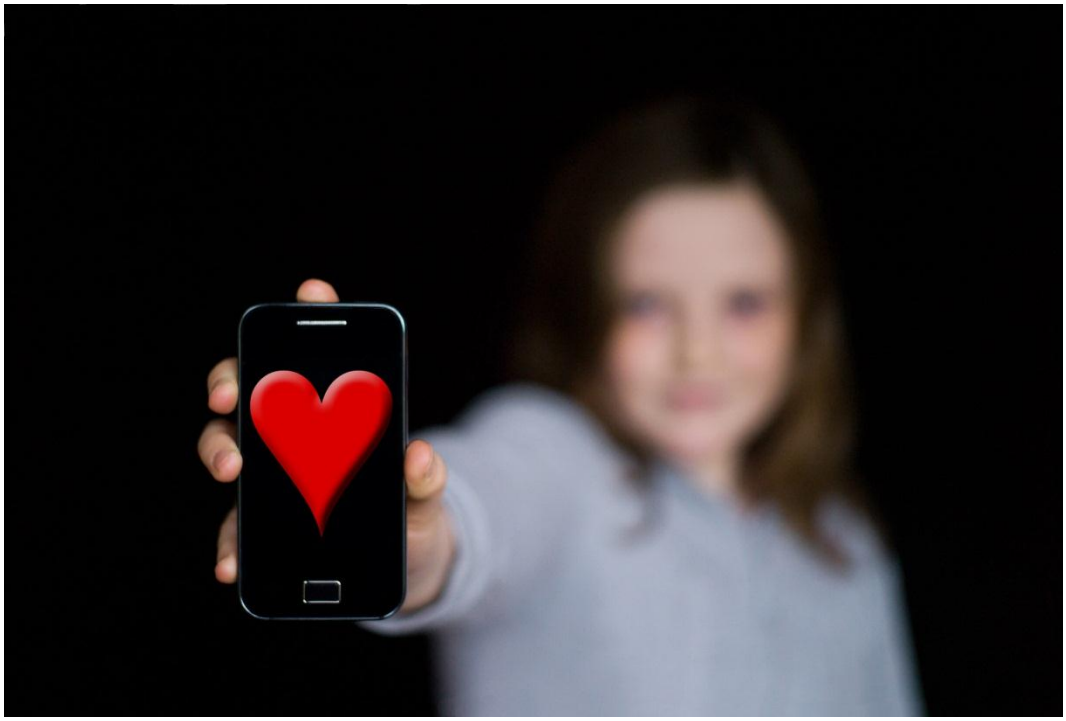
I ♥ My Smartphone



A Computing Science Course in Mobile App Development

by Jeremy Scott

TUTOR NOTES

THE ROYAL SOCIETY OF EDINBURGH

bcs The Chartered Institute for IT Academy of Computing

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, Massachusetts Institute of Technology
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 those concerned for permission to use and adapt their materials.

Contents

Overview

Introduction

Implementation of Curriculum for Excellence and the development of new National Qualifications presented a timely opportunity to revise the way Computing Science is taught in schools and to provide a more interesting, up-to-date and engaging experience for both tutors and learners.

Although first to be published, this is intended as the third in a series of resources developed by the Royal Society of Edinburgh and the BCS Academy of Computing that exemplify a subset of the Computing Science-related outcomes of CfE at Levels 3 & 4 and beyond (specifically, in this case, National 4 Computing Science).

This resource will seek to consolidate Computing concepts introduced in the previous resources through the medium of mobile app development. In addition to providing a course in programming for mobile devices, it will explore new paradigms in Computing such as mobile technologies and new interfaces, whilst providing ample opportunity for inter-disciplinary linkage.

All three resources build on state-of-the-art understanding of the pedagogy of Computing, drawn from around the world. This should enable learners to develop both app development skills *and* deep understanding of core Computing concepts and the ability to think like a computer scientist.

Whilst this resource is intended support tutors' thinking about how they might translate the intentions of the curriculum into classroom activity, it should not be seen as prescriptive. Rather, it is intended to stimulate innovation and offer tutors the flexibility and opportunity to deploy their creativity and skills in meeting the needs of learners.

Computational Thinking

Computational thinking is recognised as a key skill set for all 21st century learners – whether they intend to continue with Computing Science or not. It involves a set of thinking practices through which to view the world:

- seeing a problem and its solution at many levels of detail (**abstraction**);
- thinking about tasks as a series of steps (**algorithms**);
- understanding that solving a large problem will involve breaking it down into a set of smaller problems (**decomposition**);
- appreciating that a new problem is likely to be related to other problems the learner has already solved (**pattern recognition**);
- realising that a solution to a problem may be used to solve a whole range of related problems (**generalisation**).

Furthermore, there are some key understandings about computers:

- Computers are **deterministic**: they do what you tell them to do. This is news to many, who think of them as pure magic.
- Computers are **precise**: they do exactly what you tell them to do.
- Computers can therefore be **understood**; they are just machines with logical working.

Whilst computational thinking can be a component of many subjects, Computing Science is particularly well-placed to deliver it.

Why Mobile App Development?

For over twenty years, the desktop metaphor as exemplified by Mac OS and Windows operating systems formed the basis of most users' experience of Computing. The resultant applications became the mainstay of the ICT curriculum and informed much of what was taught.

Today's learners have a different experience of Computing: it is on-line, social and increasingly mobile. Computing devices have become more tactile and personal, the result of convergence of numerous technologies from multi-touch to motion-sensing and GPS.

This resource seeks to tap into these developments and provide an experience of Computing Science which will engage with learners in a way that is more relevant to their own digital lives.

Using this resource

As well as lessons, exercises and sample answers, this resource contains many suggested supplementary activities and inter-disciplinary learning opportunities.

It is not the author's intention that all the activities are attempted; rather, they are simply suggestions as to the kind of activities that tutors may find useful.

Feel free to use this resource as you wish:

- as part of an introduction to Computing Science within Curriculum for Excellence;
- to support the National 4 Software Design & Development unit or to form the basis of work done in the Added Value unit;
- as an introduction to Software Design & Development in the National 5 course.

Above all, this resource should not be seen as prescriptive. It merely contains guidance and suggestions as to the kinds of approach which can make learning more engaging, whilst fostering computational thinking and greater understanding of Computing Science concepts in learners.

Curriculum for Excellence outcomes

This resource seeks to address the following outcomes within Curriculum for Excellence:

- Using appropriate software, I can work individually or collaboratively to design and implement a game, animation or other application. *TCH 3-09a*
- I can build a digital solution which includes some aspects of multimedia to communicate information to others. *TCH 3-08b*
- I can debate the possible future impact of new and emerging technologies on economic prosperity and the environment. *TCH 4-01c*
- By learning the basic principles of a programming language or control technology, I can design a solution to a scenario, implement it and evaluate its success. *TCH 4-09a*
- I can integrate different media to create a digital solution which allows interaction and collaboration with others. *TCH 4-08c*
- Through research, I can gain knowledge of computer systems or emerging technologies to understand their differing features and consider their suitability for the world of work. *TCH 4-08d*
- I can create graphics and animations using appropriate software which utilise my skills and knowledge of the application. *TCH 4-09b*

App Inventor

App Inventor (**http://appinventor.mit.edu/**) is a software development environment originally developed by Google. It allows users with little or no experience of programming to create applications that can be installed and run on devices running the Android operating system. Towards the end of 2011, Google open-sourced App Inventor and its development was taken up by the Mobile Learning Lab at the Massachusetts Institute of Technology.

App Inventor uses a graphical interface, similar to that found in MIT's Scratch. In creating App Inventor for Android, Google drew upon significant prior research in educational computing which was informed by constructionist learning theories. These theories emphasise programming as a vehicle for engaging powerful ideas through active learning.

As a blocks-based programming language, App Inventor all but eliminates a major problem for learners presented by traditional text-based languages i.e. the requirement to recall and type instructions according to a strict syntax.

At the time of writing, App Inventor was still in early development at MIT. Consequently, it is strongly recommended that tutors using this pack spend some time to familiarise themselves with the environment and the Known Issues section of this resource before using it in the classroom.

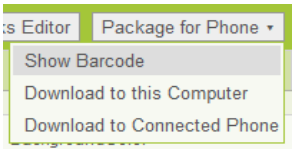
http://appinventor.mit.edu

Known Issues

- App Inventor is a cross-platform development environment (Windows, Macintosh & Linux), but creates software for Android devices only.
- All users of App Inventor require a Google account (apps are stored in the cloud). **It is recommended that tutors ensure that learners have a Google account in advance of them using App Inventor for the first time.**
    - The advantage of this is that learners are able to access their work at home.
    - Tutors may also wish to keep a copy of learners' usernames and password reminder clues (known only to the learners themselves).
- Any proxy or web filtering must be configured to allow access to the App Inventor URL.
- At the time of writing, App Inventor is best used with smartphones, not tablet computers.

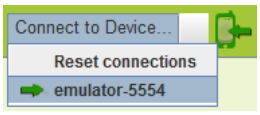
Using App Inventor in the classroom

- Whilst App Inventor offers users a software emulation of the Android operating system, best results are achieved when using an Android smartphone connected directly to the host computer for **live testing** of an application.
  This is particularly relevant when using features such as text messaging or motion sensing.
    - For live testing, devices must be in debug mode. To enable this, go to **Settings→Applications→Development→USB Debugging**
    - In the blocks editor, select **Connect to phone**
- Once an App Inventor app has been completed, there are **three** methods of getting it onto a phone:
    - By generating a QR code which provides a download URL (you will need a QR code scanner app such as QR Driod, freely available from Google Play)
    - By downloading a compiled and packaged .apk file onto a computer which can then be copied to a connected phone
    - Downloading to a connected phone via App Inventor (easiest for learners).



- It is recommended that learners **leave all the App Inventor windows open**, as restarting the blocks editor and emulator can be sluggish. If they wish to free up space on their screen they should minimise their windows rather than close them. When a new App Inventor project is chosen, all windows will update to display the new project.

- Occasionally, it is necessary to restart an app during development. To do this:
    - In the Blocks editor, choose “Connect to device” and reconnect to your device/ emulator.



    - AI will then ask:

- Use the Blocks Editor’s zoom control (opposite) to enlarge blocks when demonstrating on the screen.
- Learners can create checkpoints which effectively save successive versions of files. The use of checkpoints by learners is encouraged, especially when developing their own apps.

Installation

App Inventor requires a little preparation before using it in the classroom…

- Whilst App Inventor is primarily browser-based software, **it also requires a local installation on client machines**, both of App Inventor and the Java Runtime Environment. The latter is often already installed on computers, but do check. Installation instructions and files can be found at:
  http://appinventor.mit.edu/learn/setup/setupwindows.html
  http:// appinventor.mit.edu/learn/setup/setupmac.html
- Whilst App Inventor does support some models of phone natively, it is usually necessary to install drivers for the phone you are using.
  For this reason, it is strongly recommended that, when purchasing phones, try to **stay with the same manufacturer** (and even model), to ensure the ability to easily share phones between different client computers whilst installing just one driver across a classroom.
- The URL of App Inventor should be whitelisted on any school proxy.
- **As previously stated, users must have a Google account to log in.** Users’ projects are stored in the cloud, but can be uploaded from and downloaded to client computers from the Component Designer screen. This is recommended for backup.
- Project files for upload/download are packaged as **.zip** files. Note that this is App Inventor’s own package and not compiled .apk (Android Package files). .apk files are compiled and packaged for use on a phone and **cannot be edited in App Inventor.** It may be helpful to view App Inventor’s. zip files as **source code** and .apk files as **object code**.

Above all, tutors are strongly recommended to test the installation on a learner account before trying to use App Inventor with a class.

Useful resources

MIT's App Inventor home page: http://appinventoredu.mit.edu/

tAIR – the App Inventor Repository: http://www.tair.info/

Publishing on Google Play (formerly the Android Market)

Unfortunately, there is no direct method of uploading App Inventor files to Google Play. Besides the lack of such a feature in App Inventor itself, the .apk files which App Inventor creates are not in the required format and lack the required embedded information.

Fortunately, some free tools have been created which can convert App Inventor .apk files ready for publishing on Google Play:

- **AppToMarket** – available from http://amerkashi.wordpress.com/
- **Marketizer** – available from http://www.taiic.com/marketizer/

To publish on Google Play, you must also be a registered developer. To register as a developer, you must do three things:

- Create a developer profile
- Agree to the Google Play Developer Distribution Agreement
- Pay a registration fee ($25.00) with your credit card (using Google Checkout)

The URL for this can be found at https://play.google.com/apps/publish/signup

Clearly, the ability to market an app in the real world could be a strong motivator for learners and introduce a wealth of inter-disciplinary learning opportunities. It may be that an establishment could create a single developer licence and a designated tutor could deal with uploads, etc.

Lessons and approach

Tutors using this resource may wish to consider the following approaches:

Screencasts

In addition to a traditional booklet, this resource makes use of screencasts to deliver some of the exemplar tutorials. Tutors wishing to use them may wish to use them on a whole-class basis and/or for learners individually, stopping and starting as they need.

The rationale behind using screencasts is that learners can make quicker progress, as it's more visual and immediate. Screencasts are also a medium that learners are familiar with in their own digital lives via services such as YouTube.

As the lessons progress, it is assumed that learners have gained an understanding of how to use the component designer, then the coding interface, so the use of screencasts as "scaffolding" is reduced.

The screencasts used in this resource are available in two way:

- as movie files which can be used on individual computers or across a network
- on the "RSE/BCS Computing" YouTube channel at : http://www.youtube.com/user/RSEComputing

Deep understanding

To accompany the lessons, there are sample written and discussion tasks to enable learners and tutors to assess "deep understanding" of the Computing Science concepts. This draws upon recent findings in the *CS Principles* course at the Universities of San Diego and Glasgow. Aspects of this approach are also seen in UC Berkley's *The Beauty and Joy of Computing* course.

Traditionally, tutors have inferred the degree of learners' understanding of what they have learned from the programs they have produced; however, research has shown that this is not always a strong indicator. Consequently, consolidation via quizzes, group discussion, questioning and class work/homework should be used to enable the tutor to formatively assess the learners' understanding throughout the course, rather than simply infer this from their completed apps.

Pair programming

Collaborative learning is a cornerstone of Curriculum for Excellence and these materials encourage this, through pair programming. Pair programming can be defined as:

"… an agile software development technique in which two programmers work together at one workstation. One, the driver, types in code while the other, the observer (or navigator), reviews each line of code as it is typed in. The two programmers switch roles frequently.

While reviewing, the observer also considers the strategic direction of the work, coming up with ideas for improvements and likely future problems to address. This frees the driver to focus all of his or her attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide." **Source: Wikipedia**

Pair programming can encourage collaboration between learners, as well as making good use of available smartphones within the classroom

It is recommended that tutors explicitly advise learners to seek help from a neighbour before asking the tutor for help. Tutors will, however, understand the need to ensure that both learners are equally engaged in the work!

Suggested activities

Within these notes are suggestion about how tutors may extend. These should not be seen as prescriptive, but as possible ways to enrich a task or topic. Tutors are free to cherry-pick these, as going through all of them is likely to significantly extend the unit.

Suggested activity These opportunities are indicated by **Suggested activity** in the left margin.

Inter-disciplinary learning

Mobile app development is a "rich" medium that offers ample opportunity for inter-disciplinary learning. Within this resource are suggestions for possible inter-disciplinary activities, as well as many of the activities being inter-disciplinary in themselves.

IDL Inter-Disciplinary Learning opportunities are indicated by **IDL** in the left margin.

Introduction to Mobile Devices

This section outlines the history of telecommunications through information, written tasks and research for pupils.

It is up to tutors to decide whether to use this as an introduction or to embed the tasks within the practical lesson sequence.

A Brief History of the Telephone

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

Suggested activity **Find out about other early researchers into the telecommunications and how Bell was not the only researcher working on this (or, arguably, the first).**

Going mobile

Suggested activity **Discuss with learners why mobile communication is so popular. Why is there a need for it? What are the uses?**

- Business
- Social
- Government
- Military
- etc.

Suggested activity **Tutors who have access to some legacy mobile phones or PDAs, could provide a useful hands-on experience for learners.**

Some suggested answers are shown below.

Year	Product	Important features
1993	IBM Simon	**The first smartphone – apps included a calendar, address book, world clock, calculator, note pad, e-mail, and games.** **Touch screen** **Text entered using "predictive" on-screen keyboard or QWERTY keyboard. Price: $899**
1996	Palm Pilot	**PDA; Touch screen (with stylus)** **Office apps as well as third party** **Handwriting recognition plus on-screen keyboard**
1998	Nokia Communicator	**Smartphone that combined keyboard and screen in a "micro laptop" clamshell format.**
2002	RIM Blackberry	**Smartphone with mini QWERTY keyboard** **Introduced RIM messaging service** **Very popular in business**
2007	Apple iPhone	**User-friendly multi-touch interface** **First app store launched with 2nd generation (3G)**
2008	HTC Dream	**First Android-based smartphone**

When mobile phones became smart phones

Suggested activity **Discuss what makes a smartphone "smart".**

Suggested activity **Show video of New York University researcher Jeff Han's demonstration of multi-touch interfaces in 2006, the year before the iPhone was introduced.**

http://www.ted.com/talks/jeff_han_demos_his_breakthrough_touchscreen.html

For a desktop experience of multi-touch, show videos of Microsoft surface:
http://www.youtube.com/watch?v=6VfpVYYQzHs

http://www.microsoft.com/surface/en/us/whatissurface.aspx

Discuss with learners how quickly multi-touch has become ubiquitous and why.

Convergence: Bringing it all together

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.

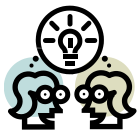
Microphone	*Function:* ***Sound input***
	Example device: ***Voice-activated device e.g. sat nav***
Loudspeaker	*Function:* ***Sound output***
	Example device: ***MP3 player***
Touch screen	*Function:* ***Touch sensing***
	Example device: ***iPod touch/nano***
Accelerometer	*Function:* ***Orientation and motion sensing***
	Example device: ***Nintendo Wii controller***
GPS sensor	*Function:* ***Geo-location***
	Example device: ***Satellite navigation e.g. Tom-Tom***
Bluetooth	*Function:* ***Radio communication***
	Example device : ***Cordless mouse***
CCD	*Function:* ***Image sensing***
	Example device: ***Digital camera***

IDL **Physics**

What is an accelerometer? How does it work? In what other devices is this technology found?

IDL **Physics**

What is GPS? How does it work? In what other devices is this technology found?



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.

Learners may use their answers from the "Going Mobile" investigation to look back a decade. By seeing what smartphones could do then, they could try to extrapolate 10 years into the future.

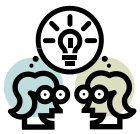
Learners tend go on flights of fancy here, so they should be encouraged to look at emerging technologies to see realistically where smartphones might be within a decade.

This could be given as a homework assignment, or learners could be encouraged to share their ideas and present to the class.

Suggested activity **To support and stimulate the task above, show learners video of Nokia Morph project** (http://research.nokia.com/morph):

http://www.youtube.com/watch?v=IX-gTobCJHs

Telephone…or computer?



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

Device	Smartphone example(s)
Input	**Touch screen; microphone; accelerometer; GPS locator**
Processing	**CPU; GPU (if present); DSP chip**
Storage	**Internal flash ROM and expansion card (if present)**
Output	**Touch screen, sound, vibration**

Suggested activity

Discuss what other devices might qualify as computers that we wouldn't automatically think of as computers.

IDL **Psychology/sociology/RMPS/responsible citizens**

Smartphones are multi-faceted communications devices that have fuelled the social networking phenomenon. What are the effects on society and individuals of always being in touch?

Discuss the pros and cons of mobile computing: health, safety, social interaction, privacy, never being out of touch, keeping up with friends/family, etc.

IDL **Language**

Class debate: *[This house believes that] smartphones have been bad for society.*

Issues to research: road casualties caused by driver/pedestrian distraction, disruption to sleep patterns due to exposure to blue light emitted by LCD screens, radiation emission, being constantly in touch with work, etc.

Contrast with the benefits of keeping in touch; increased business and convenience of shopping from anywhere; leisure/games; general convenience of converged technologies in a single handset.

Smartphone software

Go over the concept of **software** and **hardware**. Make sure learners are comfortable using the terms **program** and **software** and understand that programs are **instructions** to the computer.

Stress that a computer is a deterministic machine i.e. it will do **only as it is instructed**.

Operating system

Ensure learners understand that when they use a computer, what they are actually working with is an operating system. The particular hardware is less important.

The OS provides a means of accessing and controlling the hardware through program instructions.

Investigation

Write down the names of some mobile device operating systems and 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*
iOS	***Apple iPhone, iPad, iPod touch***
Symbian	***Nokia phone***
Windows Mobile	***Microsoft-based smartphone***
BlackBerry OS	***BlackBerry smartphone***

Apps

Investigation

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

Microsoft Office – £109 (home & learner version)

Adobe Photoshop – £657.60 (CS5, not discounted)

FileMaker Pro – £219

The Sims – £40

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

Learners could investigate Google Play or Apple's App Store here.
Any 3 here – answers will vary

Video **How much does it cost to make an app? (Appiction: App Dev 101). Show and discuss:**
http://www.youtube.com/watch?v=b3KAAdzx5VU

The mobile app industry

Video **Show video of young "bedroom developer" and discuss:**
http://www.bbc.co.uk/news/technology-16306742

Further inter-disciplinary learning

Global Citizenship

In the developing world, many countries are leapfrogging wired networks in favour of wireless networks accessed on mobile devices.

Maps of submarine cables:

http://www.submarinecablemap.com/ (clickable)
http://digg.com/newsbar/topnews/The_Internet_s_Undersea_World_Cable_Network_Map

What has been the impact on local economies that are largely driven by sole traders at markets? What role did this technology play during the Arab spring?

Responsible Citizens

Where do all the mobile phones go?
Investigate the problem of electronic waste and its disposal.

Chemistry

Smartphone touch screens are coated with Indium tin oxide, a rare earth compound that is both optically transparent and electrically conductive. Rare earths are also used several other areas of microelectronics.

Investigate rare earths:

- What are rare earths elements?
- Where are they found?
- What are some of their properties?
- What is the supply situation like?

Maths

Finnish company Rovio, creators of Angry Birds, spent approximately 100,000 euros developing the game. It went on to make over 50 million euros in its first year alone.[1]

Imagine that you were given £100,000. Using a spreadsheet or other tool, analyse the tasks involved in bringing a product to market and create a budget. Consider all the jobs involved in creating a successful game and how you might divide up the sum to cover all the costs.

[1] Source: http://www.wired.co.uk/magazine/archive/2011/04/features/how-rovio-made-angry-birds-a-winner

Lesson 1: Virtual Pet

Concepts introduced

- App Inventor environment with concepts of **components**, **properties** and **code**
- Event-driven programming
- Touch interface

Computational Thinking themes

- Abstraction
    - what causes things to happen (event) : button click, canvas drag, accelerometer shake
    - when things happen (action following event)
    - what happens e.g. sound plays, sound vibrates
    - position represented x & y coordinates
- Pattern recognition
    - same behaviours for cat & dog in Extension task
- Algorithm
    - sequence
    - event triggers action

Objectives

Learners should be able to:

- label the major parts of an App Inventor program
- understand how the components and blocks work and interact
- understand the difference between the user interface and the program code

Outline

- Engage: Demonstration of VirtualPet app
- Explore: Learners create VirtualPet app
- Explain: Learners individually annotate App Inventor handout
- Elaborate: Modify VirtualPet app
- Conclusion

Materials

- VirtualPet apps: **VirtualPet1**, **VirtualPet2**
- Screencasts: **VirtualPet1**, **VirtualPet2**
- Handouts: **App Inventor Handout** (see Appendix C)

Introduction

This is the Android equivalent of a "Hello world" program and serves as an introduction to the App Inventor environment.

- Demonstrate a fully-functional version of the app (File: **VirtualPet1**) to learners. For this you can use a projector connected to a computer with the app running on the Android emulator or display the app working on a phone (either installed or live testing).
- Ask learners to predict what you should do and what will happen. Then show them that if you touch the picture, the phone makes a noise.

Task 1: Getting started with App Inventor

- Learners should then load up screencast **VirtualPet1**, which will introduce them to the App Inventor environment and assist with logging on, etc.

Task 2: Creating code

- Once learners have done this, they should open screencast **VirtualPet2**, which will take them through creating the app themselves.
- Stress the difference between **My Blocks** (blocks available for your app's components) and **Built-in blocks** (general programming structures)

Task 3: Testing your app

- Help learners debug their app.

Task 4: App Inventor reference sheet

- Issue **App Inventor Handout** (see Appendix C) and get learners to fill it in, annotating what they think the parts of App Inventor do. After they have finished, collect the handouts and use them for formative assessment, to determine what learners already understand.
- Pass out unmarked copies. Go through the answers with the learners, having them write in the correct answers. Use the sample answers to guide the discussion, but allow learners to offer their ideas, making sure they write down the correct answer.
- Ensure learners understand the difference between the **interface** and the **program**.

Learners should retain the handout for future reference.

Extension task 1

- Replace the button with a **Canvas component** (Basic→Canvas), which has the **DrawingCanvas.Dragged** block to detect dragging, rather than simple touch. This allows simulation of stroking the cat, rather than simply touching it.

Extension task 2

- Add an empty **Image sprite component** (Animation→ImageSprite) under the cat's chin. Drag over this item to make the cat purr – i.e. for the phone to vibrate, using a **Sound.Vibrate** block – when it's stroked under its chin.
- Does the cat meow as well as purr when you stroke under its chin?
  Discuss with learners what this tells us about how the image sprite interacts with the canvas.

Did you understand?

a) Try out the app and describe the problem.

App produces the same sound for both animals.

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

Set appropriate sound source in each event before playing it (Set <sound component>.source to<filename> block)

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

Show learners what happens when set sound source block is inserted AFTER the sound is played. Get them to predict this.

Restart app and ask learners to predict what will happen when the Cat button is clicked; then the Cat button again; then the Dog.

The sound may be right to start with, but will be wrong when a different button is chosen. Ask learners why.

In this way, learners can be shown how the sequence of the code is vital to achieving the correct output.

Once learners get program working, let them experiment with it.

NB They can use App Inventor's Checkpoint facility (**Checkpoint** button in Component Designer) to save their work at various points.

Conclusion

- Revise **components**, **properties** and **code**.
- Ask learners to summarise what they saw and learned.
- Emphasise that the **order of execution** is vital when programming.
  Stress that computer will do **only what it is instructed/programmed to do**.
  If an app doesn't work as expected, then we have made a mistake!

Further extension work

- Show how to amend the program to create buttons to change the picture and sound for different animals (see **VirtualPet2**). Learners could use such an extension to create a simple application to help very young children recognise well-known animals.
- Use this app as the basis of a soundboard or virtual instrument e.g. a drum kit or piano, using empty sprites as overlays on drums, keys, etc.

Lesson 2: Finger Painting

Concepts introduced

- Touch screen interface
- Variables
- Procedures
- Interface design
- Event-driven programming: events and event handlers
- Maintain a running total variable

Computational Thinking themes

- Abstraction
    - variable name/value corresponding to data in memory
    - variable types: integer (brush size & line width); image
- Pattern recognition
    - same behaviour for different colour button clicks
    - similar behaviours for size button clicks
- Algorithm
    - small & big size changes
- Decomposition
    - general colour event breaks down to specific red/green/blue event
    - general size change breaks down to small/big change

Objectives

Learners should be able to:

- explain the role of the component designer, block editor, and phone/emulator
- understand the role of problem decomposition - that it is important to break a larger problem into smaller parts and solve one part at a time
- understand the event-driven nature of App Inventor programming
- understand that variables contain values and make programs easier to modify later

Outline

- Engage: Demonstrate working FingerPaint app
- Explore: Learners go through FingerPaint tutorial
- Explain: Tutor provides help as needed
- Evaluate: Tutor & learners check that apps work
- Elaborate: Tutorial extensions
- Deep understanding: Did you understand?

Materials

- Apps: **FingerPaint1**, **FingerPaint2, FingerPaint3**
- Screencasts: **FingerPaint1**, **FingerPaint2**

Introduction

Display FingerPaint app (**FingerPaint1**) to learners.

Ask learners to predict what you should do to make it work and what will happen when you do; then demonstrate the app. Whilst demonstrating the app, discuss the interface:

- Why are icons used?
- What are the advantages/disadvantages of using icon buttons instead of text buttons?
- What effect does it have on the overall impression you get of the app?

Either show to whole class or let learners watch individually screencast **FingerPaint1** which demonstrates creating the app's interface.

Task 2: Creating the code

Point out that while we have a good interface, there is no functionality. Get learners to run the app in the emulator to emphasise this.

Demonstrate problem decomposition by going over the algorithm with learners. Show how you were able to deal with the overall problem by focussing on small parts of it separately to make it more manageable i.e. "Divide & Conquer".

Display **algorithm** and **code** for this app on the board and discuss how they are related. Explain that algorithm and code are **two representations of the same thing** –an approach to solving the problem. This is clear example of levels of **abstraction**: the problem, the algorithm and the final code.

Stress that writing an algorithm allows you to work out the stages the program will have to go through to solve the problem without worrying about the code.

Task 3: Fixing the Bugs

Introduce idea of programming errors. Stress that computers are **deterministic**, so that if code doesn't work as expected, it's because the programmer has made a mistake.

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:** **Create a new button for black paint (and code it)**

b) **Line width:** **Have more choices for line width; have a "slider" or other control that lets you set a variable line width.**

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

- **Design code carefully by writing down the algorithm in advance**
- **Trace through the algorithm before coding, trying to work out what the computer would do, in order to catch errors in your design.**

Variables

The use of variables is a key Computational Thinking concept (abstraction) and important for learners to understand. A variable can be explained as a part of computer memory in which we store a value. We then label it, like a box, so that we can remember what's stored inside.

Suggested Activity

Pass out envelopes with names written on the front (such as "age", "favourite colour" "pet", etc.). Inside each one, have a piece of paper with a value written on it. Take out each value and put in a new one to demonstrate assignment of a new value to the variable.
Use this activity to discuss the importance of creating meaningful variable names so that we can recognise them easily in our program.
Use this activity to demonstrate data types i.e. we can't assign a value for colour such as "red" to an age variable, etc.

Ask learners:

Q: Why store information in variables? Isn't it more complicated?

A: No! Variables make it **easier** for us to use and change information in a program. Explain how variables allow us to **generalise** our code.

Some examples:

- Consider a game that displays a score which changes throughout the game.
  Would we have a separate program line for every possible score in a game? No, we'd have one program line, but refer to the variable. A variable is the only real way of storing this information.
- Consider a program that calculates prices by adding on VAT.
  Without variables we would have to set the value of VAT as a number in every calculation in the program. If the rate of VAT changed, we would have to go through

our code, changing the number every time we had used it.
By using a variable to store the rate of VAT, we would only need to change the value assigned to the variable once.

Extension 1: More flexible brush size

Explain the common practice of changing value of variable by referring to itself
i.e. new value of variable = old value of variable +/- amount

This can be done by asking a learner to mentally add up a series of small numbers as you provide them, then give you the final total. Ask the learner: How did you do this? You maintained a running total, rather than remember all the individual numbers and add the together at the end.

Demonstrate to learners adding a label to display the brush size variable (app **FingerPaint3** features this)

Suggested activity

This resource does not go into parameter passing explicitly; however, there is clearly scope to introduce it here, especially for learners who have used procedures with parameters in environments such as Snap!/BYOB. Such learners may see redundant code which could be generalised using a single procedure to change brush size, with a parameter/argument of +1 or -1 passed into it accordingly.

Extension 2: Any colour you like

This may require resizing of components, depending on the resolution of the phone, but will make the app more useful

Extension 3: Cool feature

Note that this feature will only work on a phone, not the emulator.

Extension 4: Another cool feature

Shows learners how to save files permanently.

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.

Did you understand (part 2)?

2.3 What **type** of variable – **text** or **number** – would the following values be stored in:

a) 23 – Number
b) Alice – Text
c) 3.14 – Number (despite point)
d) SG12 RDW – Text (despite digits)
e) Fourteen – Text

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?

It makes it difficult to identify what value each variable is storing – and therefore more difficult to correct bugs.

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)

player1; player2; player1score; player2score

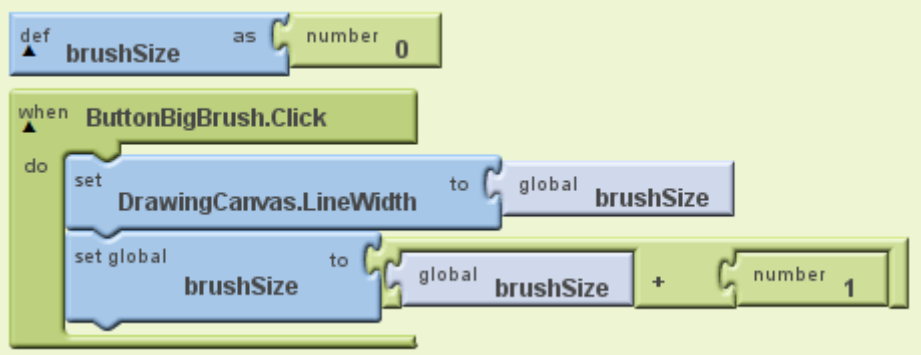
2.6 Think of some non-computing examples of "variables" and their possible values. An example is shown below:

Variable: cutlery Possible values: knife; fork, spoon

Variable: gender
Possible values: male, female

Variable: musicalInstrument
Possible values: guitar, drums, piano, bass, flute

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



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: The brushSize variable is initialised at 0. When ButtonBigBrush is clicked, the canvas LineWidth is set to the current value of variable brushSize – in this case zero – then the brushSize variable is incremented.
The lineWidth would therefore still be set to zero until ButtonBigBrush is clicked again. Even then, it would always be one size "behind" the button clicks.

Correct code:

```
def brushSize as number 0
when ButtonBigBrush.Click
        set global brushSize to global brushSize + 1
        set drawingCanvas.LineWidth to global brushSize
```

Conclusion

1. Explain the role of the **component designer**, **block editor** and **phone/emulator**.
   - What does each do?
   - How does adding a component affect the block editor/phone/emulator?
   - How does creating a block of code change the designer/phone?
   - Does interacting with the phone change the program code?

2. Why did we use variables for brush size instead of "hard coding" it?

   To make it easier to modify later. Using variables *generalises* the problem of creating a brush size.

3. What is a bug? What different kinds are there?

   An error in code which stopes it from working as expected
   e.g. syntax error; logic error.

Lesson 3: MoleMasher game

Concepts introduced

- Problem decomposition and modularisation using procedures
- Timers/clock component

Computational Thinking themes

- Abstraction
    - repetition
    - position: x & y coordinates
    - variable
- Algorithms
- Decomposition
    - use of procedure to move mole
    - use of procedure to update score
- Pattern Recognition
    - use of one procedure being called repeatedly to update score

Objectives

Learners should be able to:

- understand the use of an algorithm to develop a solution to a problem
- become more familiar with translating an algorithm into code
- become more familiar with using variables
- use procedures to create a more modular program

Outline

- Engage: Demonstrate working MoleMasher app
- Explore: Learners go through MoleMasher tutorial
- Explain: Tutor provides help as needed
- Evaluate: Tutor & learners check that apps work
- Elaborate: Tutorial extensions
- Deep understanding: Did you understand?

Materials

- App: **MoleMasher**
- Screencasts: **MoleMasher1**, **MoleMasher2**

Details

- Demonstrate a fully-functional version of the app (**MoleMasher**) to learners. For this you can use a projector connected to a computer with the app running on the Android emulator or display the app working on a phone (installed or live testing).
- Ask learners to predict what you should do and what will happen. Write this out as Input, Process and Output
- Go through algorithm and try to elicit as much of the code from learners as possible without using the screencast.
- Learners will likely need explanation of why we subtract the width & height of the mole from the canvas width and height when generating random coordinates for the mole to appear.

Pair programming

- Assign learners to pairs. Explain pair programming - one person is the "driver" and does the clicking and typing. The other person is the "navigator" who has the screencast and tells the “driver” what to do. Learners will swap roles every 5-10 minutes.
- Tutor should keep track of the time and announce that learners should trade places at even frequencies. Monitor that both learners are contributing equally.
- Tutor should walk around and help pairs who are stuck as needed. If pairs complete tutorial, they should save then begin an extension with a new file name.

Procedures

Emphasise to learners how procedures let us break a problem down into smaller problems and solve each one separately.

- Stress that building a program from smaller “sub-programs” is a very important idea in Computing Science. In the software industry, software is often written by teams of programmers. By breaking down a problem into smaller tasks, it lets different people or groups within the team works on separate parts of the program at once.
- For example, if a company were creating a word processing application, one group of developers might work on the spell checker, another on the printing features, another on the help screens and so on…
- Remind learners that a procedure is a "sequence of statements that you can refer to as a single instruction" i.e. it’s like creating a new block.

Using the **MoleMasher** app’s **UpdateScore** procedure, explain why it’s more efficient to define a procedure once and call it up repeatedly, rather than have the same code appear several times in a program (Computational Thinking – abstraction).

- Q: Why?
  A: Because if you need to change the code, you only have to change it in one place.

Q: Where have we seen this idea already?
A: Variables

- Go back to the FingerPaint app and ask learners to look for repeated code that could be made into a procedure.

Link in with comments here, pointing out to learners that they should always **comment** a procedure to explain what it does. Remind them that comments are for **programmers**, not users (a common misconception).

Did you understand?

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

Change the interval property for the game timer, so the mole appears more quickly.

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

a) Getting ready for school

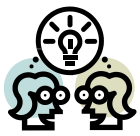
Wake up, breakfast, washed, dressed, gather things, leave house

b) Making breakfast

Get bowl, get milk, empty cereal into bowl
Boil kettle, put bread in toaster, put teabag in pot, etc.

Note that this is one of the stages in part a), showing that successive decomposition can take place. Therefore one procedure can be broken down into further procedures, etc.

In the case of tea & toast, this could even be used to introduce the concept of parallel processing i.e. whilst waiting for the toaster to pop up or the kettle to boil, we get the other ingredients, for example.



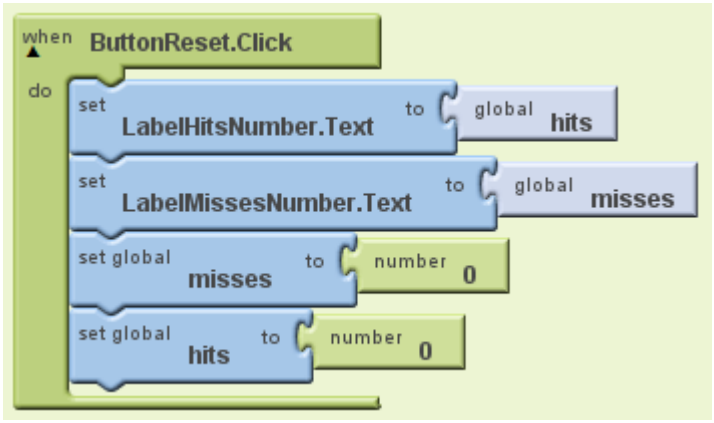
3.3 Discuss with your partner some examples of sub-tasks within a simple "space invader"-type game that could be coded as procedures.

Write them in the space below.

Move ship, move aliens, update score, game over, etc.

Stress that there are clear stages in the game that can be worked on separately.

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



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

LabelHitsNumber: 10 ; LabelMissesNumber: 5

The values have remained the same because the variables have been initialised after setting the text values.

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

when ButtonReset.Click do

set global hits to 0

set global misses to 0

set LabelHitsNumber.text to global hits

set LabelHitsNumber.text to global hits

Lesson 4: Times Table Helper

Concepts introduced

- Further problem decomposition using procedures
- Fixed loops and loop index/counter

Computational Thinking themes

- Abstraction
    - data type: Boolean
    - selection: if
- Decomposition
    - break down creation of table into procedures for header and body
- Pattern Recognition/Abstraction
    - individual lines of each table created by repeated formula with variables
- Algorithm/Abstraction
    - repeated formula for lines of table used again in weight & currency extensions

Objectives

Learners should be able to:

- understand the use of an algorithm to develop a solution to a problem
- become more familiar with translating an algorithm into code
- become more familiar with using variables
- use procedures to create a more modular program
- understand the use of fixed loops and a loop variable

Outline

- Engage: Demonstrate working TimesTable app
- Explore: Learners go through TimesTable tutorial
- Explain: Tutor provides help as needed
- Evaluate: Tutor checks that apps work
- Elaborate: Tutorial extensions
- Deep understanding: Did you understand?

Materials

- App: **TimesTable**
- Screencast: **TimesTable**

Details

- Demonstrate a fully-functional version of the TimesTable app to learners. For this you can use a projector connected to a computer with the app running on the Android emulator or display the app working on a phone (installed or live testing).
- Ask learners to predict what you should do and what will happen. Write this out as Input, Process and Output
- Go through algorithm and try to elicit as much of the code from learners as possible without using the screencast.
- Explain the use of a FOR loop to repeat a fixed number of times. Stress to learners that this kind of loop requires that we **know in advance** how often a task is repeated. If they have experience of an environment such as Scratch, compare it to a REPEAT block.
- Dry run (on the board or on paper) a FOR loop e.g. repeat printing a sentence 5 times. Then introduce the use of the loop variable to number the sentences.
- Once learners have done this, go through the Times Table Creator algorithm with them. Points to note:
    - Q: How will we store the table number?
      A: Use a variable
    - Q: What are the main stages in this app?
      A: Get the table number, create the header and create the table.
    - Q: what would be the best way of dealing with the separate stages of creating the header and creating the table?
      A: Use procedures.
- Dry run on the board
    - building up the text for each line of the table using both text blocks and variable values
    - then appending these to the label's existing text.
      Suggestion: link with incrementing a numeric variable in FingerPaint and MoleMasher app.
- Link in with comments here, pointing out to learners that they should always **comment** a procedure to explain what it does.

Extension 1

Stress to learners that this app requires no input from the user, but just uses the loop variable and the multiplier of 2.2 to create a table.

Extension 2

Show the similarity between the **TimesTable** app and this one – it's essentially the same app, with a different multiplier variable.

www.x-rates.com is a good exchange rate web site.

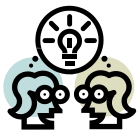
Learners could also add a suitable background image to the screen component.

Did you understand (part 1)?

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.

Try it if you haven't already noticed this (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).



4.1 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?

It generalises the problem (abstraction): using a variable allows us to deal with any times table– and to use the same code to do so. The alternative would be to have specific code for each times table (and obviously only permit a limited choice of tables).

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

The Times Table label's text property isn't initialised; the app therefore continually appends the new times table to the old one.

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

Set the text property of the Times Table label to an empty string at the start of the CreateTimesTable procedure.

4.4 After discussing your answer to 4.2 (above) with your tutor, make the change to your code. Did it fix the problem?

A common mistake is that learners will try to initialise the property at the end of the CreateTimesTable procedure, thereby "blanking out" the table that was just created. It will seem to learners as if nothing is being displayed at all.

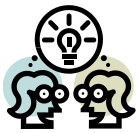
Did you understand (part 2)?

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**.

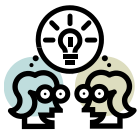
Don't crash and burn!

An app which crashes will attract very few users!



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

There is no value for the CreateTimesTable procedure to multiply the loop counter with.



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

Prevent the app from continuing until it gets a [numeric] value into the text box.

Validating input

- Link into the fact that we have already tried to limit the range of possible inputs to numbers by selecting the NumbersOnly property in the TextBox component.
- Once learners have created a Notifier which validates the input, ask them if it is still possible to enter an invalid input. If so, what further measures could we take to prevent the app from proceeding until it has a valid input?

Did you understand (part 3)?

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

a) someone's age: **Numbers 0–122**[2]

b) the number of days in a month: **1-31**

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

Zero.

4.8 In the previous example, 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 learners' 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): **C, D**

[2] Jeanne Calment, the oldest verified human, died in Arles, France (the place of her birth) aged 122 years and 164 days on 4th August, 1997.

Lesson 5: Virtual Map Tour

Concepts introduced

- Lists as data structures
- Linking to external data services

Computational Thinking themes

- Abstraction/Pattern Recognition
    - use of corresponding lists to hold related data
- Decomposition
    - break down problem into noting list selection index and selecting equivalent item from related list

Objectives

Learners should be able to:

- understand the use of an algorithm to develop a solution to a problem
- become more familiar with translating an algorithm into code
- become more familiar with using variables
- use procedures to create a more modular program
- understand the use of fixed loops and a loop variable

Outline

- Engage: Demonstrate working MapTour app
- Explore: Learners go through Map Tour tutorial
- Explain: Tutor provides help as needed
- Evaluate: Tutor checks that apps work
- Elaborate: Tutorial extensions
- Deep understanding: Did you understand?

Materials

- App: **MapTour**
- Screencast: **MapTour**

Details

Demonstrate a fully-functional version of the **MapTour** app to learners. For this you can use a projector connected to a computer with the app running on the Android emulator or display the app working on a phone (installed or live testing).

Lists

Introduce the idea of storing related data in a list with learners, then build up to the idea of tying multiple lists together with elements in equivalent locations.

Q: Where else could we use this?
A: For example, in a high score table, with names and scores stored in equivalent locations of lists.

Extension 1

Learners can make this as local as they want – to their county, town or even street.

Extension 2: Cool feature

The Location Sensor has many blocks related to GPS such as longitude, latitude, altitude, etc. Learners could create a live readout of any of these properties.

Further extension

IDL **Geography**

Learners could create a geo-caching/treasure hunt app, with clues tied to locations – either in their local area, or virtually, with clues visible from Google's Street View.
If not set locally, the app's locations could be within a country/city they are studying in geography, or even a world-wide hunt around famous locations.

Lesson 6: Heads I Win

Concepts introduced

- Conditional loops (and contrasting with fixed loops)

Computational Thinking themes

- Abstraction
    - fixed repetition: FOR loop
    - conditional repetition: WHILE loop
- Decomposition
    - use of procedure to initialise
    - multiple stages within loop
- Pattern Recognition
    - use of same procedure for button click and accelerometer shake
- Algorithm
    - counting length of run; reset initial value if run broken

Objectives

Learners should be able to:

- understand the use of an algorithm to develop a solution to a problem
- become more familiar with translating an algorithm into code
- understand the use of variables to maintain a count within a conditional loop

Computational Thinking themes

- Abstraction (through the use of variables)
- Pattern Recognition (through the use of one procedure being called repeatedly)
- Algorithms
- Decomposition

Outline

- Engage: Demonstrate HeadsIWin app
- Explore: Learners go through HeadsIWin algorithm and create code from this
- Explain: Tutor provides help as needed
- Evaluate: Tutor checks that apps work
- Elaborate/ Deep understanding: Did you understand?

Materials

- App: **HeadsIWin**

Details

Demonstrate a fully-functional version of the HeadsIWin app to learners. For this you can use a projector connected to a computer with the app running on the Android emulator or display the app working on a phone (installed or live testing).

Go through algorithm and try to elicit as much of the code from learners as possible without using the screencast.

This lesson is the first time learners will be required to create code from an algorithm without the support of a screencast. In recognition of this, there is nothing in the development of the app that that the learners have not been exposed to already (other than the WHILE block).

Conditional Loop

Go over the idea of a conditional loop. Try to elicit situations where it could be used.

Do a dry run through the algorithm on the board, asking learners to pick a random value for the side variable. Maintain a trace of the variables and show how the test in the loop is applied every time. Keep this on the board for the extension tasks.

The event-driven nature of App Inventor forced the use of this tried-and-tested exemplar for a conditional loop, as events cannot be handled inside one.

Extension 1

Learners will need to introduce a variable (e.g. tossCounter) to maintain a count of the total number of tosses. Common mistakes include:

- Forgetting to declare and initialise tossCounter variable
- Incrementing the variable at the same point as incrementing headsCounter (inside the IF statement)
- Forgetting to initialise tossCounter this every time ButtonTossCoin is clicked
- Forgetting to append the final value of tossCounter to the end of the output.

Extension 2

Learners need to work out the condition that ends the run i.e. a tails, and at this point initialise headsCounter to 0 inside the **else** statement.

The output will surprise learners, especially in terms of its variability. It can be an effective warning against the perils of gambling!

Extension 3

Introduce a new variable (tailsCounter).

Learners will need to work out that this is an extension of the previous example, whereby each side will re-initialise the counter for the other side i.e. a heads will result in tailsCounter = 0 and tails in headsCounter = 0.

It will also introduce a complex condition into the loop i.e.
WHILE headsCounter = <6 OR tailsCounter<6

Q: Ask learners what would be the effect of changing the OR to an AND?

Q: Ask learners what would be the effect of changing the code to:

WHILE headsCounter = <0 OR tailsCounter<0

or **WHILE headsCounter = <0 AND tailsCounter<0**

This could provide an opportunity to introduce the idea of an **infinite loop**. Contrast this with a fixed/FOR loop and why the latter can't be infinite.

NB If an app performs a lengthy operation, Android will block the app's user interface until the code has finished. In these circumstances, Android will display an "Application not responding" (ANR) dialogue if an activity does not react within 5 seconds. From this dialog the user can choose to stop the application, so choose any further work in this respect with care!

Extension 4: Cool feature

Learners should recognise that they can create a **procedure** to toss the coin which can be called in both the button click and accelerometer shaking events. This reinforces the idea of procedures allowing us to **write once and call many times**.

NB Stress to learners that giving the phone any more than a gentle flick will cause the app to at least hang (as it tries to process lots of shake events) or even crash.

Lesson 7: Wiff-Waff Game

Concepts introduced

- Collision detection
- Working with graphics
- Motion detection and control

Computational Thinking themes

- Abstraction
    - bat and ball properties
- Decomposition
    - ball position determines action
    - ball in same position as bat/edge/bottom
    - use of procedures for set up; increase score; lose game
- Algorithm
    - determine ball's position and act
- Pattern Recognition
    - multiple use of procedure for increase score
    - reinitialise game state after losing

Objectives

Learners should be able to:

- understand the use of an algorithm to develop a solution to a problem
- become more familiar with translating an algorithm into code
- understand the use of a timer (integral to ball component) to trigger events in a program to maintain a count within a conditional loop

Outline

- Engage: Demonstrate working **WiffWaff** app
- Explore: Learners create **WiffWaff** game
- Explain: Tutor provides help as needed
- Evaluate: Tutor and learners check that apps work
- Elaborate/ Deep understanding: Did you understand?

Materials

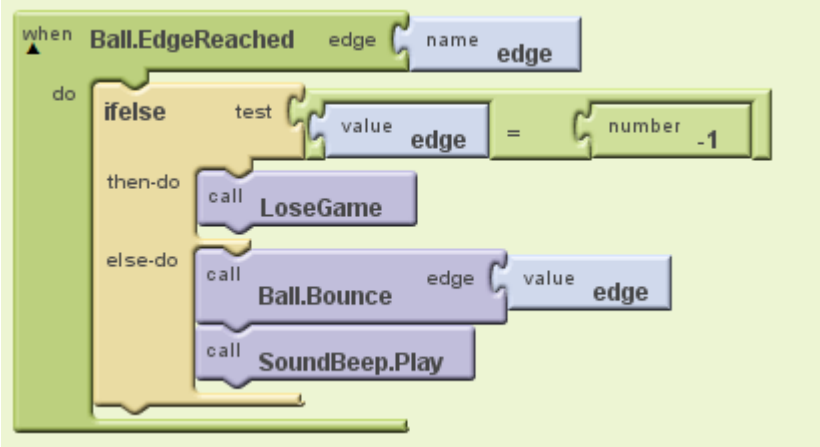
- App: **Wiff-Waff**

Details

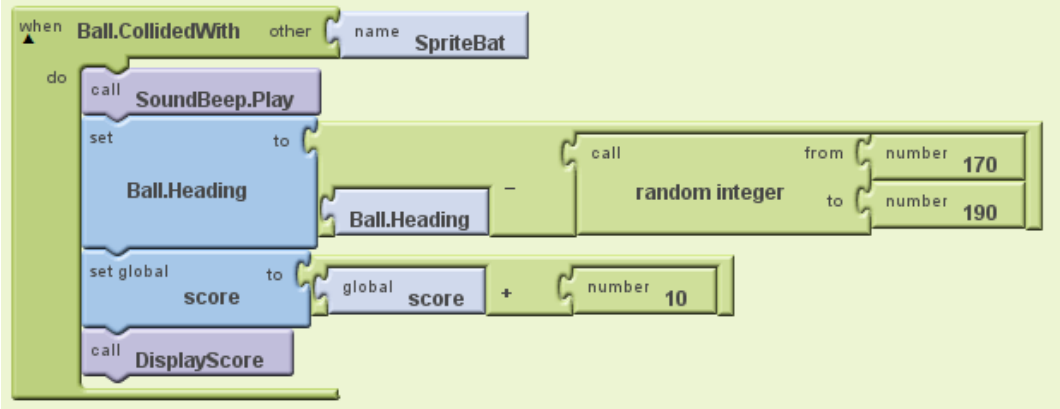
Whilst this app requires more code than learners have previously produced, stress to learners that they should concentrate on only one part at a time. By having the code split into events or procedures, the task of creating the app is therefore made much easier.

Point out to learners that the interval on the ball acts like a built-in clock/timer for this component.

Learners are likely to require help coding the **when ball reaches an edge** stage of the algorithm. This makes use of the **when Ball.EdgeReached** block w ith edge as a parameter.



For bat collisions, learners should use the **Ball.CollidedWith** block, with their bat sprite as a parameter (below). Once again, it is likely that they will require help with this.



Did you understand (part 1)?

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

It is impossible to restart the game.

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

Add a line to enable the start button.

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.

It makes it easier to locate and correct errors in code. It also allows us to add to a sub-task by making the changes in one place only i.e. it generalises the problem.

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

Because there is only one condition to test for which simplifies the code.

Extension 1: Skill levels

This will involve a certain amount of rearrangement of the components. Learners may also wish to consider altering the scoring for different levels.

Further Extensions

- Include a TinyDB component to save the previous score.
- The author's own students particularly enjoyed recording their own sounds, especially for the LoseGame procedure. A taunt of "Wiff-Waff!" seemed to go down best!

Did you understand (part 2)?

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)
```

7.3 What direction will the bat move if the phone is completely level?

It will move to the left.

Why?

Because the roll value will be zero, which falls within "else" in the algorithm.

Mobile App Project

Analyse

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

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.

Encourage learners to consider the factors in bold (above). They may even wish to include a user persona to help clarify this (below):

User Persona Example for app which sends friends a wake-up text
Jo
14 years old
Has to get up early for school (on weekdays) and a paper round (on weekends, and holidays)
Tends to oversleep
Has received a warning for punctuality slipping below 90%
Owns an Android smartphone

Now discuss your ideas with your tutor.

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

The main issue to look out for here is to ensure that the scale of the project is achievable for the learners concerned.

Design (Interface)

Make a sketch of your app's interface
Your sketch should be labelled to show what each component does.

Allow learners to create wireframes of their app. Ensure these are appropriately annotated.

There's a good series of App 101 videos from Appiction on YouTube:

Video **Why design is important: http://www.youtube.com/watch?v=BuP29Zmglks**

Video **Wireframing your app: http://www.youtube.com/watch?v=nCmV-8xA48M**

Design (Code)

Stress to learners that they should take time to get their algorithms right before coding. Remember the ancient proverb: hours of coding can save minutes of design!

Implement

Remind learners about use of meaningful identifiers for components, variables and procedures. Remind learners about the importance of internal commentary in code.

Test

Test your app to make sure it works.

Learners should perform self-testing and get their classmates to test . The may wish to issues questionnaires to their test group.

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!

Learners should show consideration of their target market here.

Evaluate

Encourage learners to discuss and reflect honestly on their work.

Maintain

Now imagine that you are adapting your app to make it work on a **tablet computer**. What changes would you make?

Brainstorm with learners aspects of tablet computing they might consider e.g.

- **screen size**
- **is the app suited to a less portable device**
- **will it benefit from rich graphics**
- **does it requires a 3G connection (not standard on tablets), etc?**

Some apps will therefore translate better to a tablet format than others.

Inter-disciplinary learning in the project

Art & Design

Design an icon, splash screen or game character for your app.

Music

Create a soundtrack or jingle for your app.

Business education/art/media studies/languages

Create an **international marketing campaign** for your app. Assign roles to different members of the team. Post it onto an app store and sell it for real – even if it's just to friends & family (see "Publishing on Google Play", page 16).

Video **How to create revenue your app (Appiction App Dev 101)**
http://www.youtube.com/watch?v=RxIPArYs8eI

Video **At what price should I sell my app? (Appiction App Dev 101)**
http://www.youtube.com/watch?v=iZUkjSmzY3c

Language

Make a **foreign language versio**n of your app. This will mean translating the marketing text and any on=screen text into the language you are studying. This is an important job in the global software industry and is known as **localisation**.

Congratulations!

Encourage learners to continue their app development at home.

Point out that they have only scratched the surface and that there are many more functions in App Inventor such as text messaging and social networking (e.g. Twitter) integration to name just two.

Appendices

Appendix A: Learner Tracking Sheet

Name: _____ Class: _____

Stage	Progress * (D, C or S)	Date completed	Comment
Introduction			
Lessons			
1: Virtual Pet			
2: Finger Painting			
3: Mole Masher			
4: Times Table			
5: Virtual Map			
6: Heads I Win			
7: Wiff-Waff			
Project			
Analysis			
Design			
Implementation			
Testing			
Documentation			
Evaluation			
Maintenance			

PROGRESS *

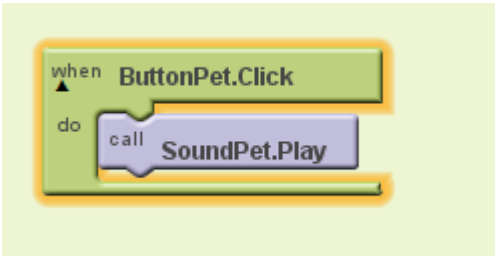
Developing where the learner is working to acquire skills or knowledge

Consolidating where the learner is building competence and confidence in using the skills or knowledge

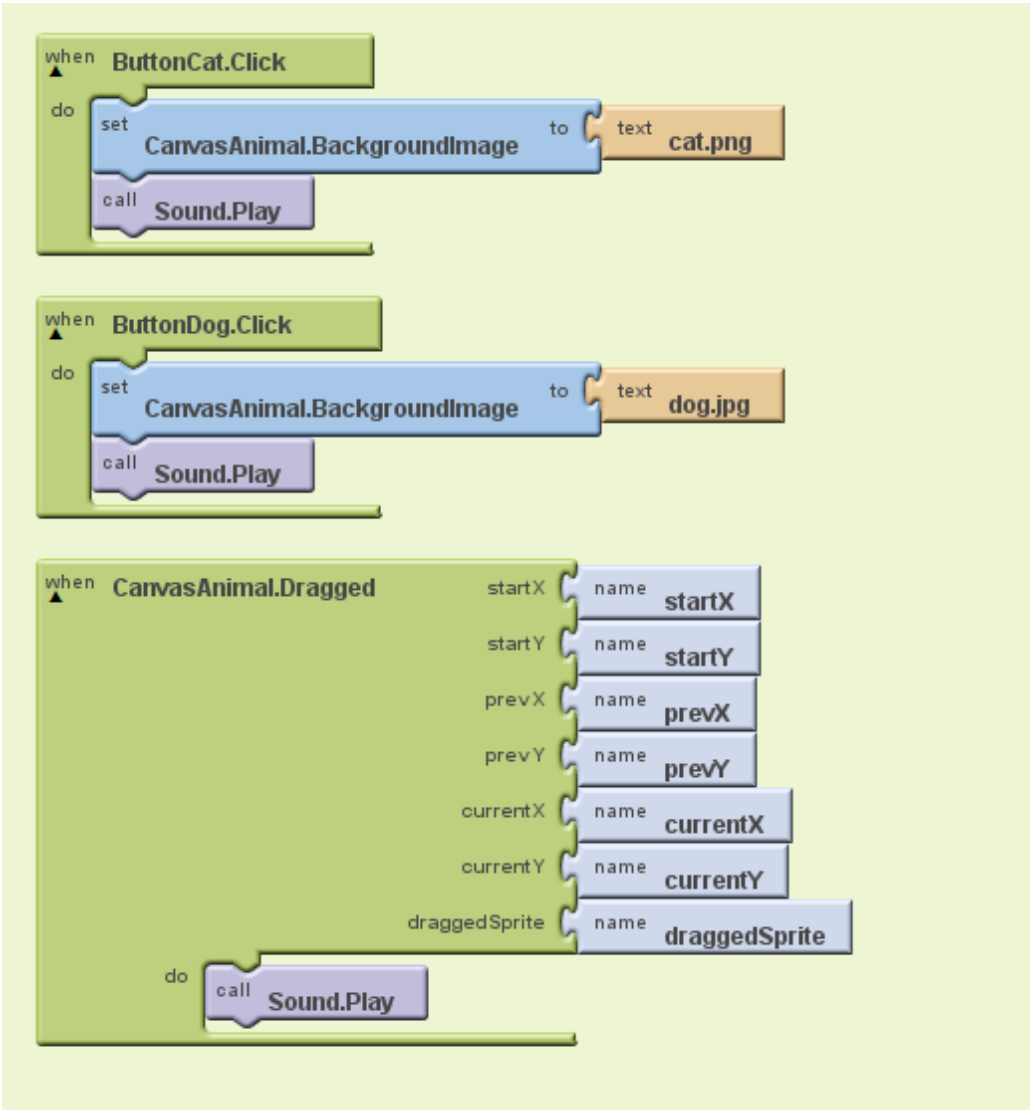
Secure where the learner is able to apply the skills or knowledge confidently in more complex or new situations

Appendix B: Sample Code

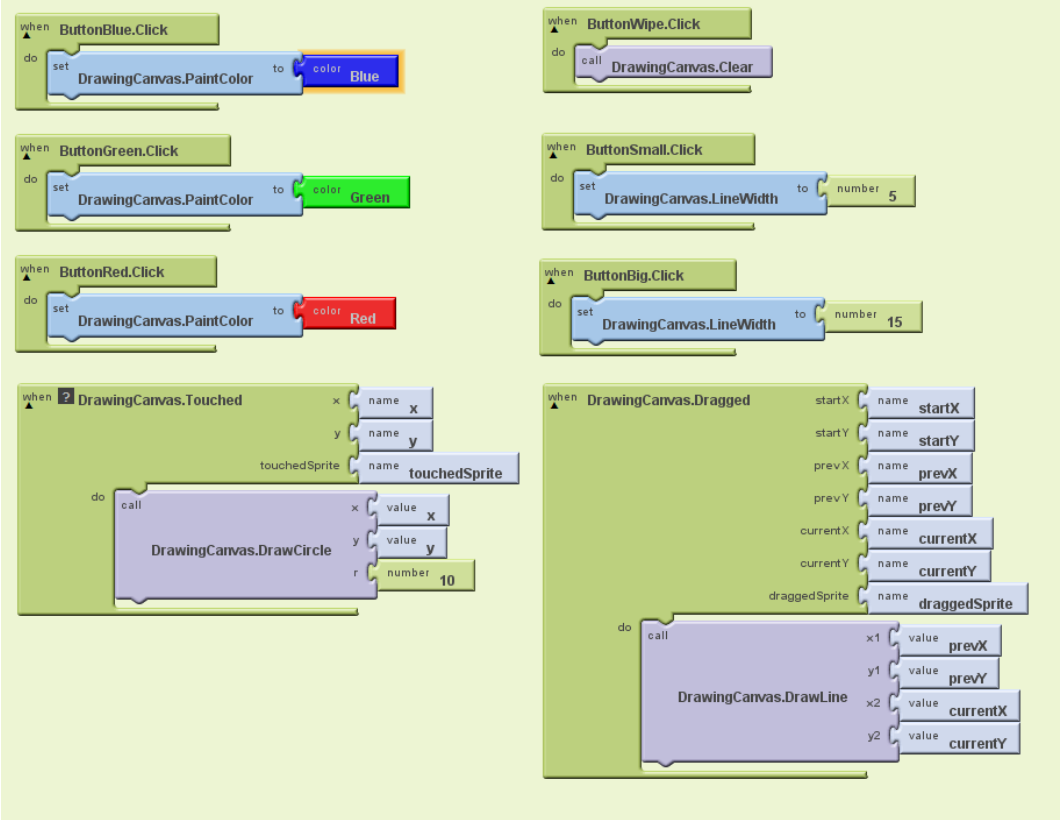
Lesson 1: VirtualPet1



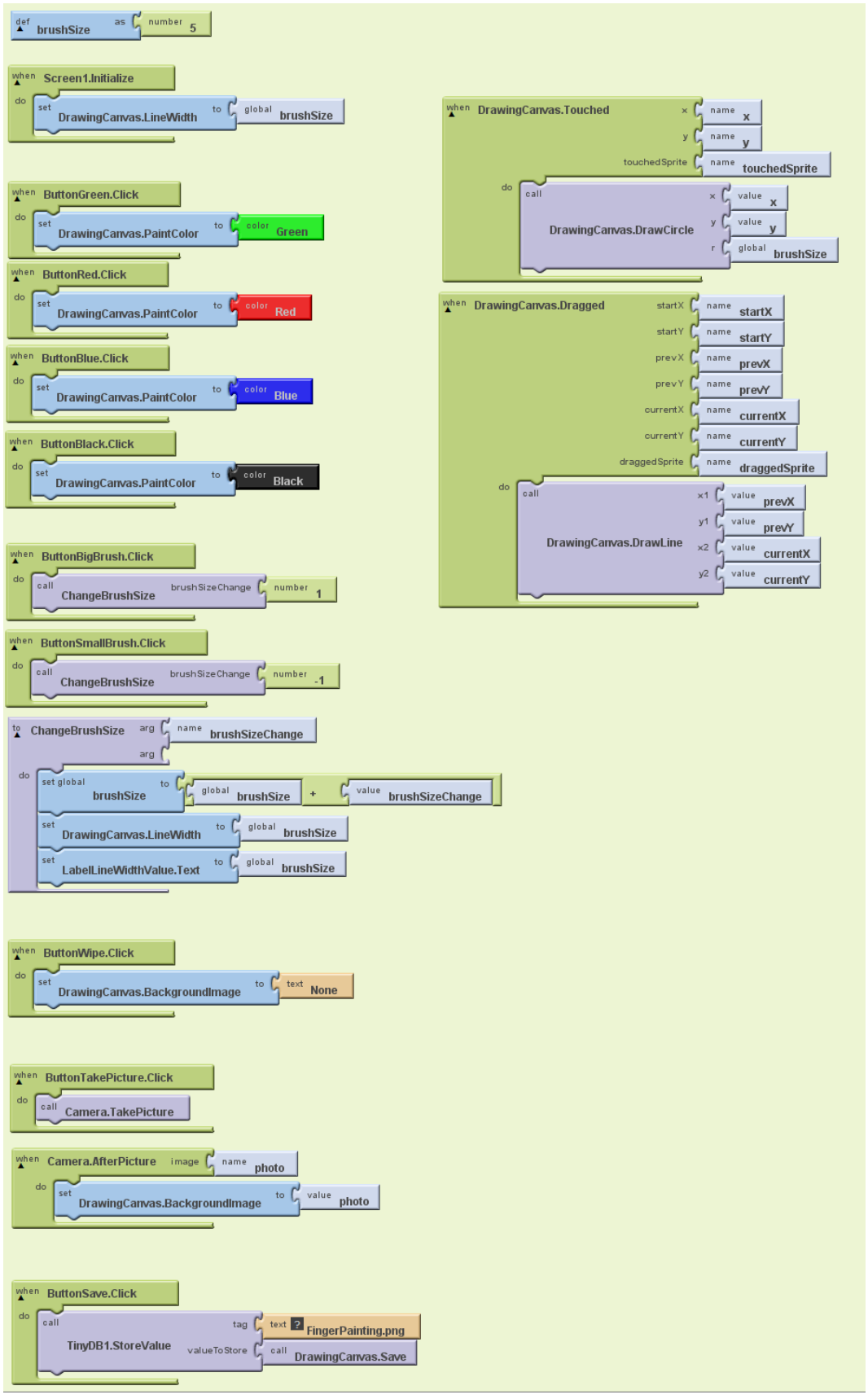
Lesson 1: VirtualPet2



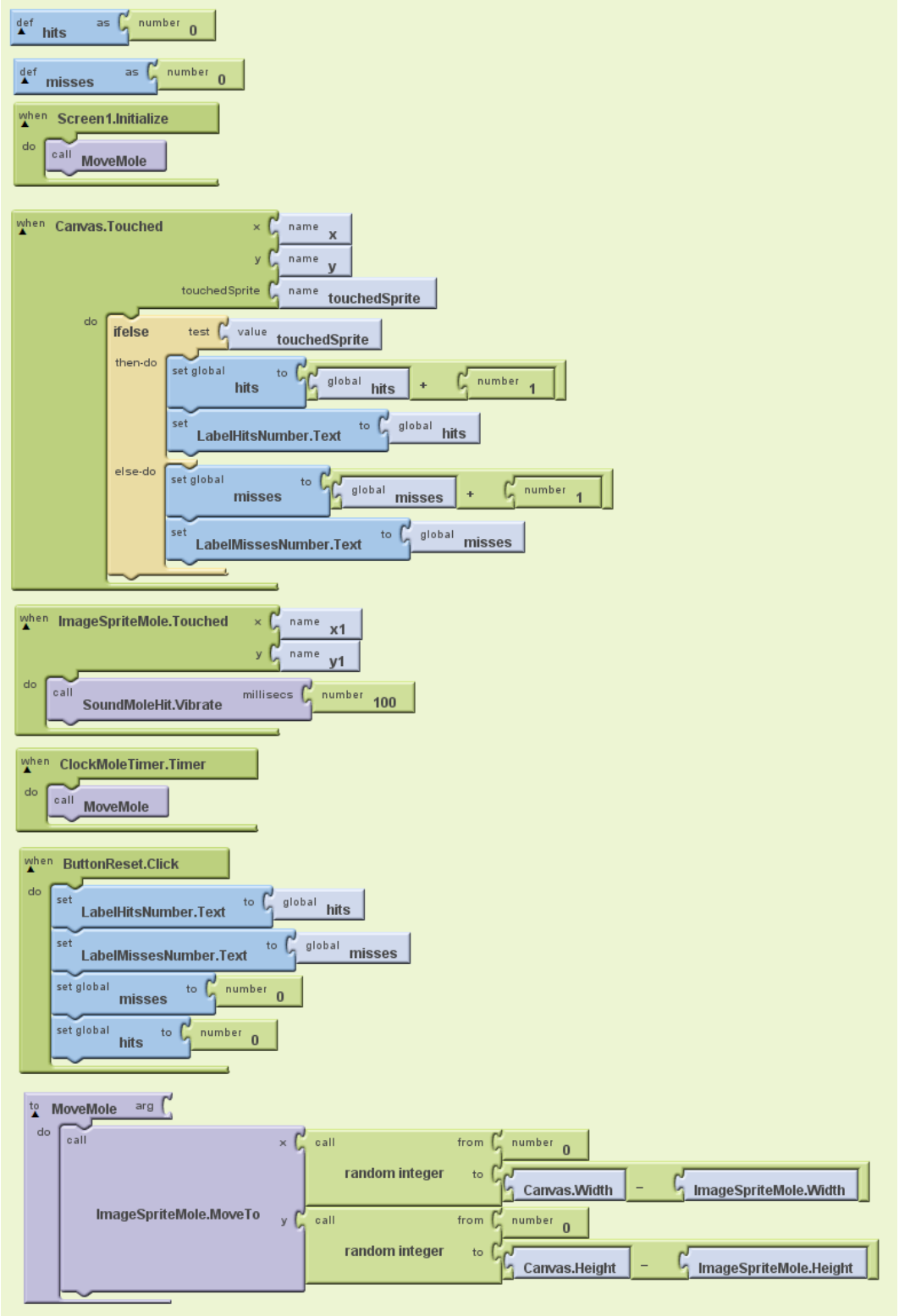
Lesson 2: FingerPaint1



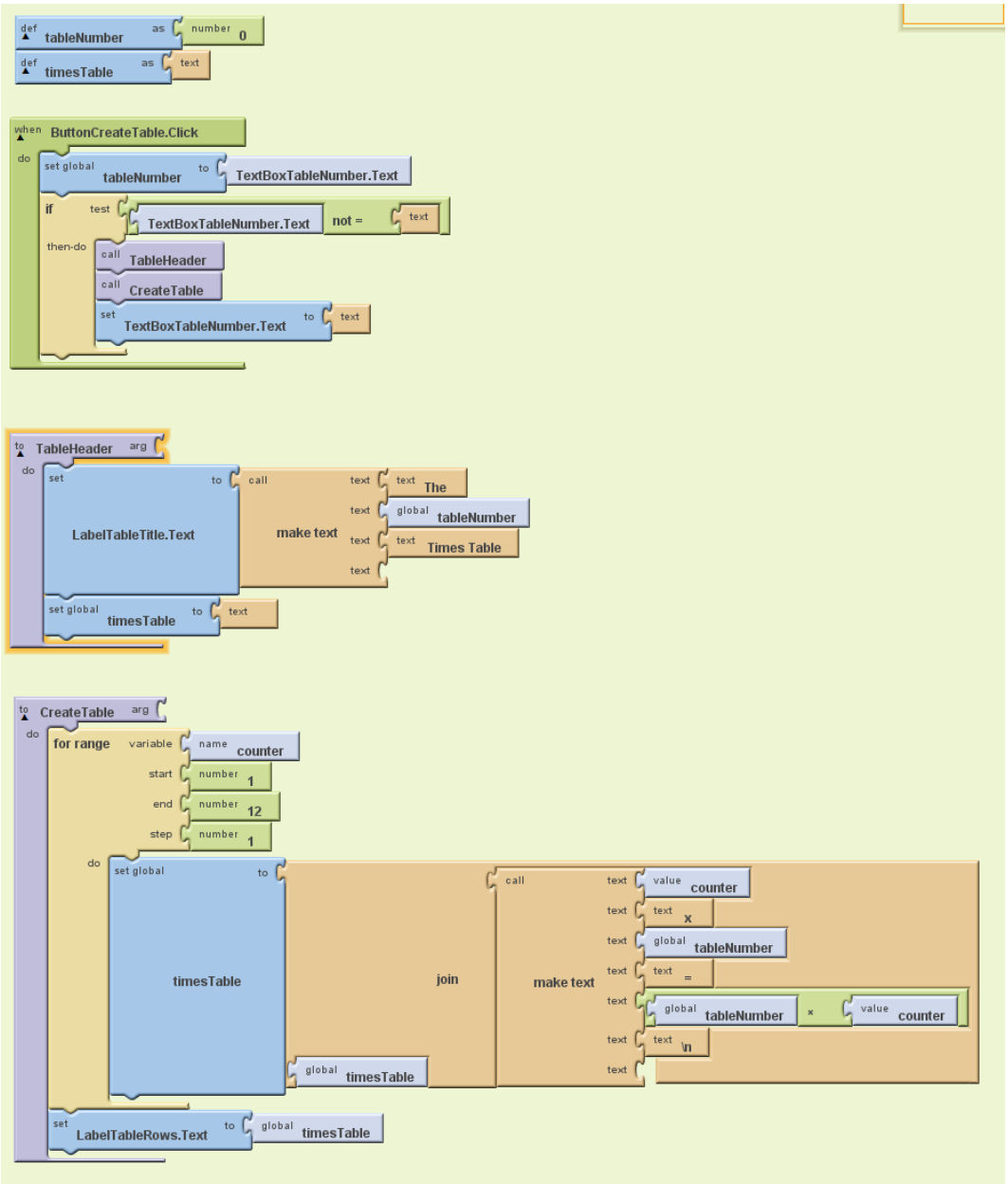
Lesson 2: FingerPaint2



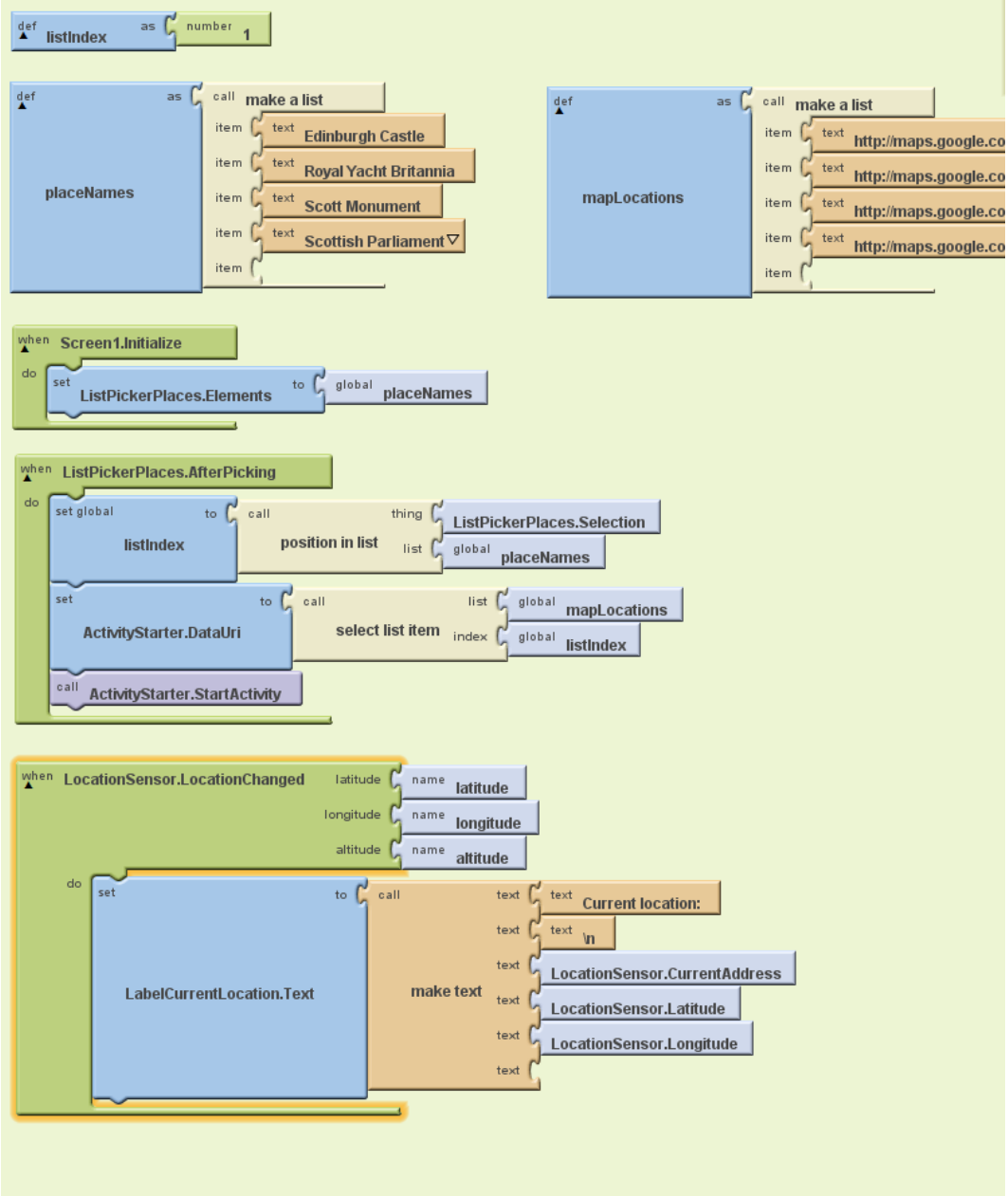
Lesson 3: Mole Masher (inc. extensions)



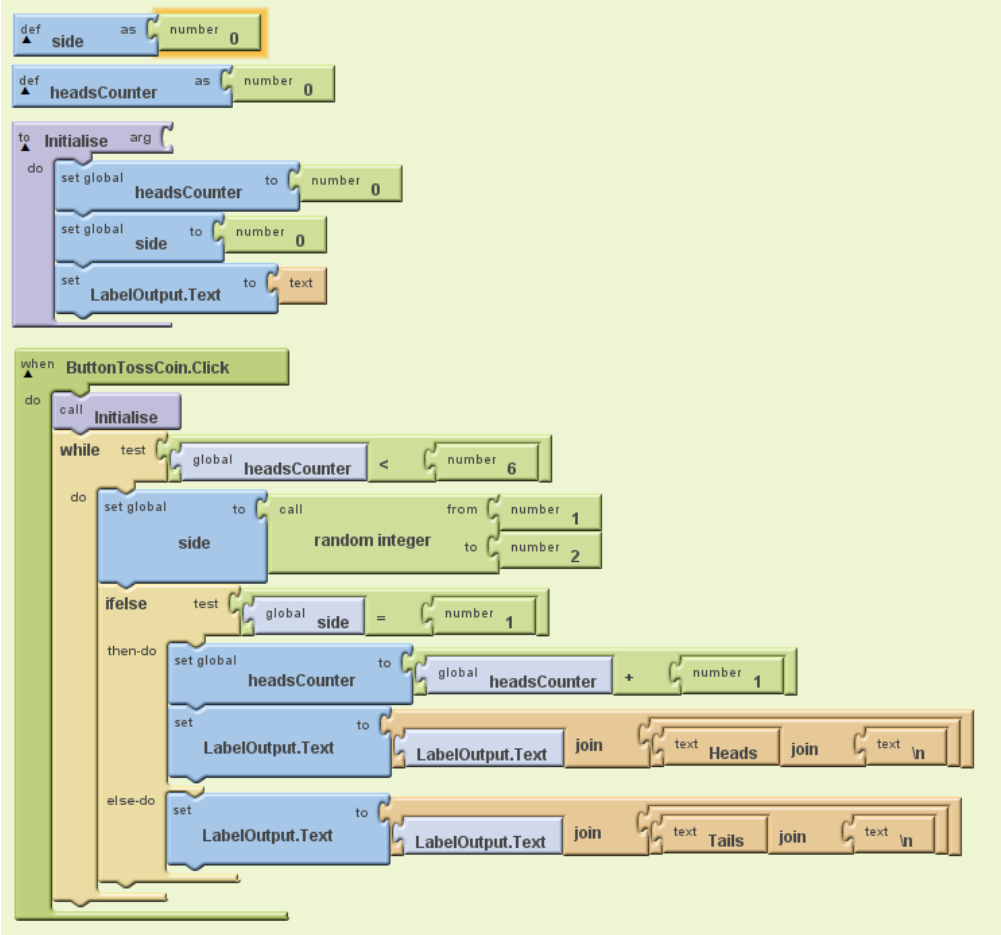
Lesson 4: TimesTable Helper



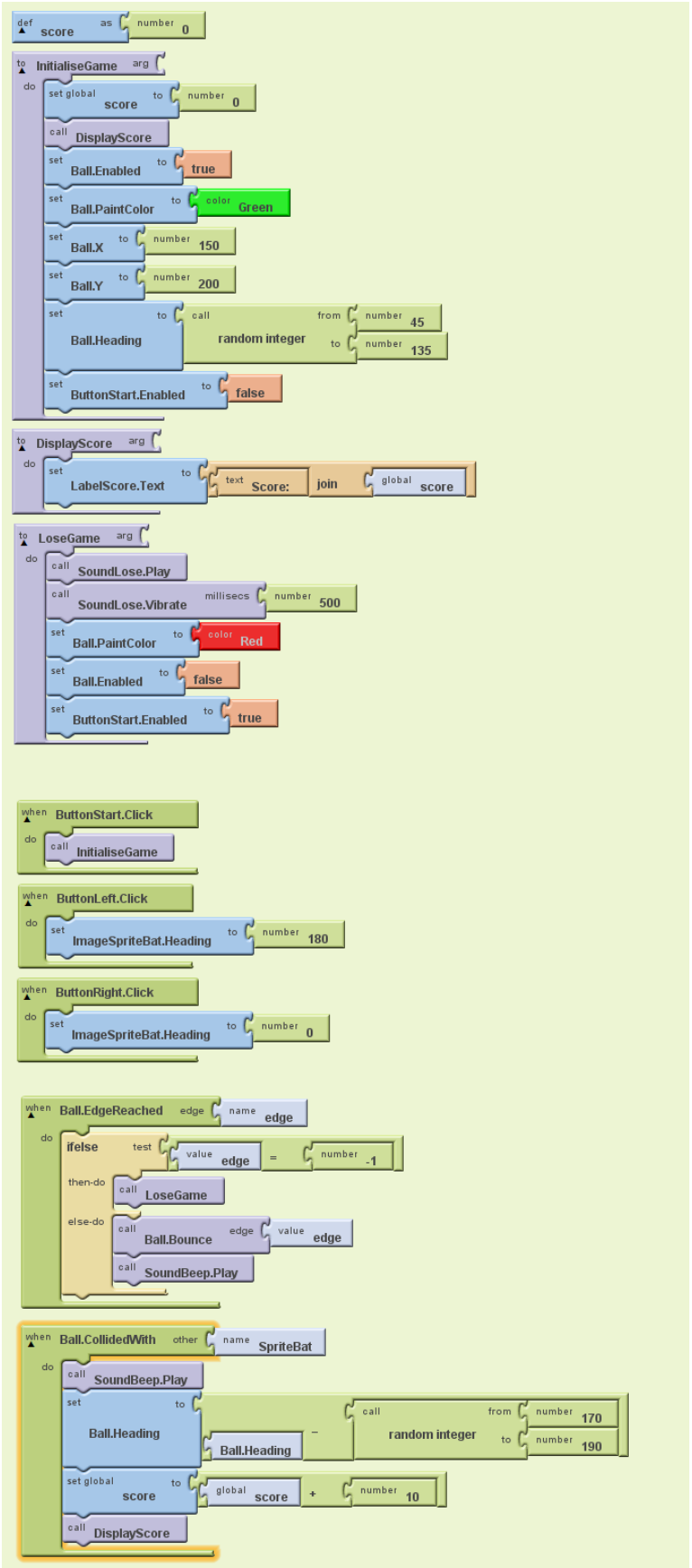
Lesson 5: Virtual Map Tour (inc. extensions)



Lesson 6: Heads I Win



Lesson 7: Wiff-Waff Game



Appendix C: App Inventor Handout

The following four pages contain:

- the **App Inventor Handout** used in Task 4 of Lesson 1 (see page 21);
- sample answers for above.

Name: _____ **Class:** _____ **Date:** _____

Mobile App Development: App Inventor

The screenshots below are from an App Inventor program called VirtualPet. When the program is running on a phone, when you "stroke" the cat by clicking the picture, it meows.

Answer the questions below.

1. PHONE

a. Explain how the Virtual Pet app works from the user's point of view:

_____

b. What happens if the user touches the picture of the cat?

_____

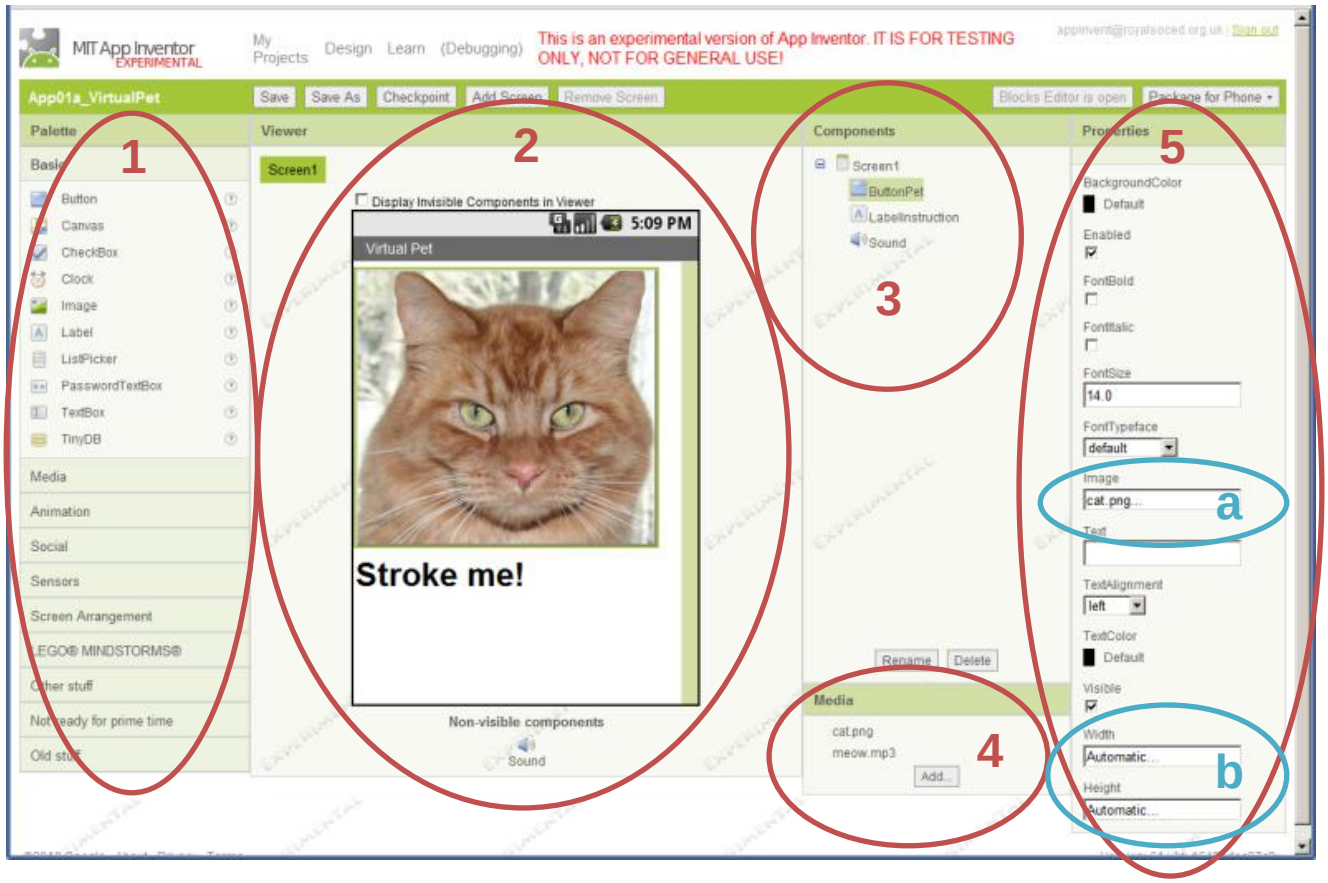
c. What happens if the user touches the "Stroke me!" text label?

_____

d. What happens if the user presses a button on the actual phone?

_____

2. COMPONENT DESIGNER



1. **Palette**: holds the components you can use in your program, separated into categories
2. **Viewer**: lets you arrange components on a rough preview of the phone screen
3. **Components**: lists components in the app
4. **Media**: lists media files for the app
5. **Properties:** of selected component

3. PROPERTIES (for button shown on page 1)

a. What does this mean about the image property for the button? What is it telling the computer?

_____

_____

b. What does this mean about the width and height for the button? What is it telling the computer?

_____

_____

PROPERTIES (for label shown opposite)

c. What does this mean about the background colour of the label?

_____

_____

d. What does this mean about the text of the label?

_____

_____

e. What does this mean about the alignment? Is the label centred?

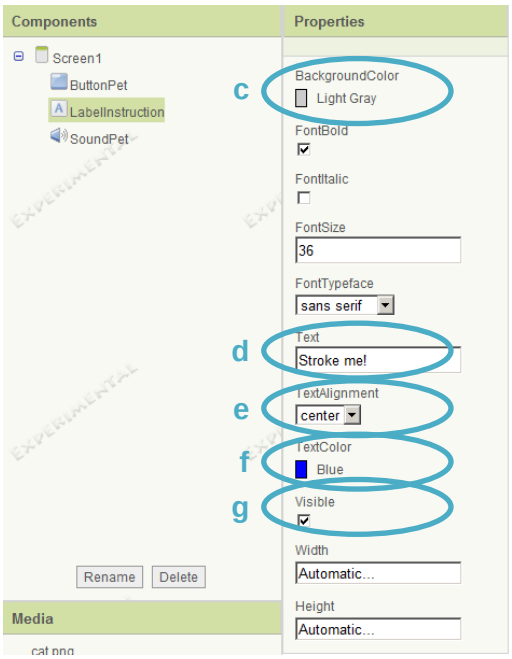
_____

_____

f. What does this mean about the text colour? What is blue?

_____

_____



g. What does this mean about the label? What would happen if it were unchecked?

_____

4. BLOCKS EDITOR

a. Explain what the example program opposite does:

_____

_____

_____

b. What is the event that causes something to happen?

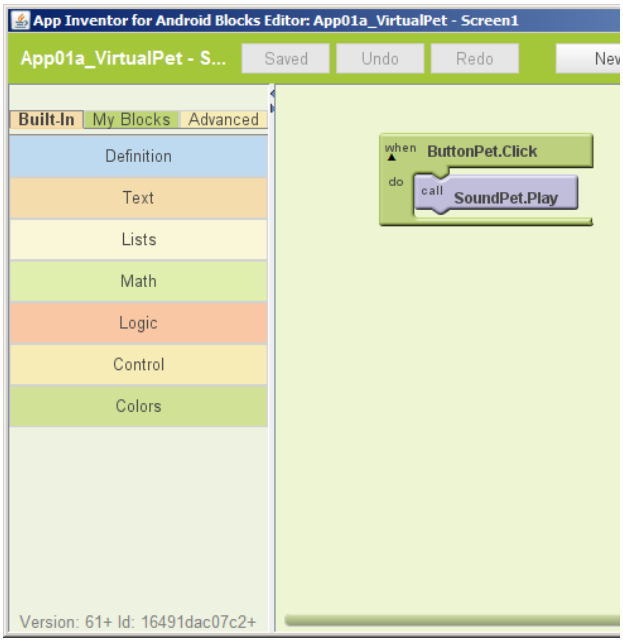
_____

_____

c. What is the action that happens after the event?

_____

_____



Name: _____ Class: _____ Date: _____

Mobile App development: App Inventor (Sample answers)

The screenshots below are from an App Inventor program called VirtualPet. When the program is running on a phone, when you "stroke" the cat by clicking the picture, it meows.

Answer the questions below.

1. PHONE

a. Explain how the Virtual Pet app works from the user's point of view:

The user strokes the cat by touching the picture of the cat. The phone makes a meowing noise.

b. What happens if the user touches the picture of the cat?

The phone "meows" by making a purring sound

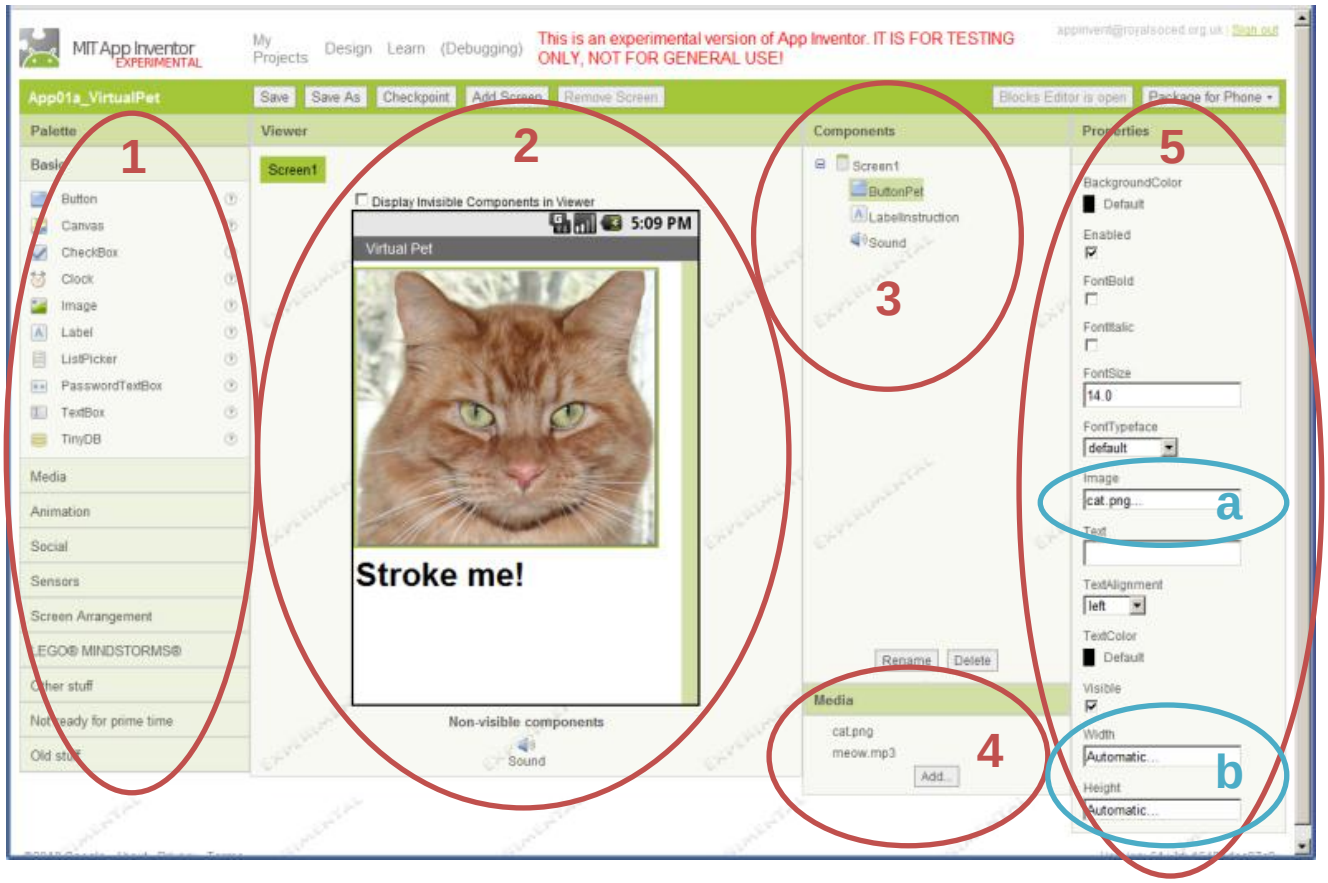
c. What happens if the user touches the "Stroke me!" text label?

Nothing

d. What happens if the user presses a button on the actual phone?

The phone does whatever that button means, like returning to main menu

2. COMPONENT DESIGNER



1. **Palette**: holds the components you can use in your program, separated into categories
2. **Viewer**: lets you arrange components on a rough preview of the phone screen
3. **Components**: lists components in the app
4. **Media**: lists media files for the app
5. **Properties:** of selected component

3. PROPERTIES (for <u>button</u> shown on page 1)

a. What does this mean about the image property for the button? What is it telling the computer?

The button looks like the cat.png image rather than looking like a plain button

b. What does this mean about the width and height for the button? What is it telling the computer?

The width and height are set automatically by the phone, based on the image size. The size will be just enough and no more to accommodate the image.

PROPERTIES (for <u>label</u> shown opposite)

c. What does this mean about the background colour of the label?

The background colour of the label is light grey. The light grey fills the whole label area, surrounding the text

d. What does this mean about the text of the label?

The label displays the text "Stroke me!"

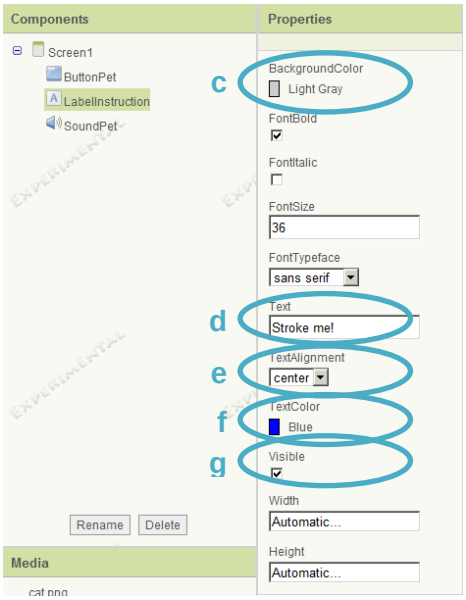
e. What does this mean about the alignment? Is the label centred?

The text is centred within the label; the label is not centred in the screen.

f. What does this mean about the text colour? What is blue?

The text is blue.

g. What does this mean about the label? What would happen if it were unchecked?



The label is visible. If the box were unchecked, the label would be invisible and hidden from view.

4. BLOCKS EDITOR

a. Explain what the example program opposite does:

When the button is clicked, the sound plays whatever it is set to.

b. What is the event that causes something to happen?

The event is when the click/press of the button.

c. What is the action that happens after the event?

The sound plays, which means that the sound file associated with the "SoundPet" component is played.