MIT App Inventor 2

Andrea Sterbini – sterbini@di.uniroma1.it

App Inventor 2: building simple Android apps

Built with Blockly http://ai2.appinventor.mit.edu

Build, compile, and deploy Android App on the phone

NEW!!! for iPhones ALSO!!!

Automatic deploy of changes while editing, either to the Phone or to an Emulator

- Install AI2 Companion App
- Run the Companion and connect by QR or code

Apps can be Packaged and installed stand-alone on the phone

Special tricks

Use an emulator instead than a phone

Genymotion for Windows, MAC or Linux

Note: in Genymotion install the Arm Translation Toolkit

BlueStacks for Windows or MAC (faster)

BEST: share phone screen on PC with scrcopy (via ADB debug)

via USB or Wifi (if your phone allows it)

The server can be LOCAL to avoid network problems

App Inventor 2 Ultimate [2018]

(or you can compile and run it from http://appinventor.mit.edu/appinventor-sources)

Web-based GUI editor

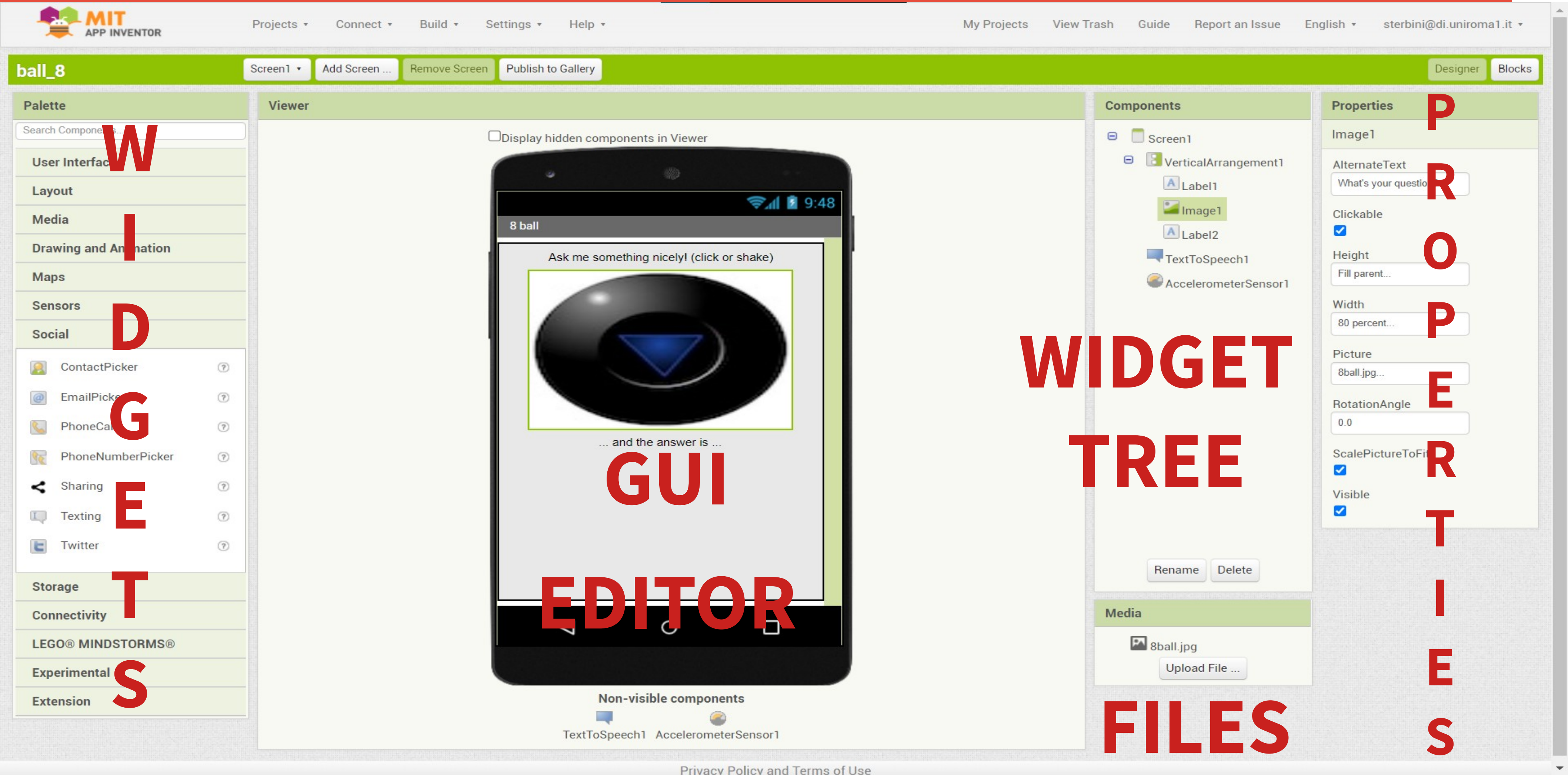


Image1

AlternateText: What's your questio

Clickable: ☑

Height: Fill parent...

Width: 80 percent...

Picture: 8ball.jpg...

RotationAngle: 0.0

ScalePictureToFit: ☑

Visible: ☑

WIDGETS

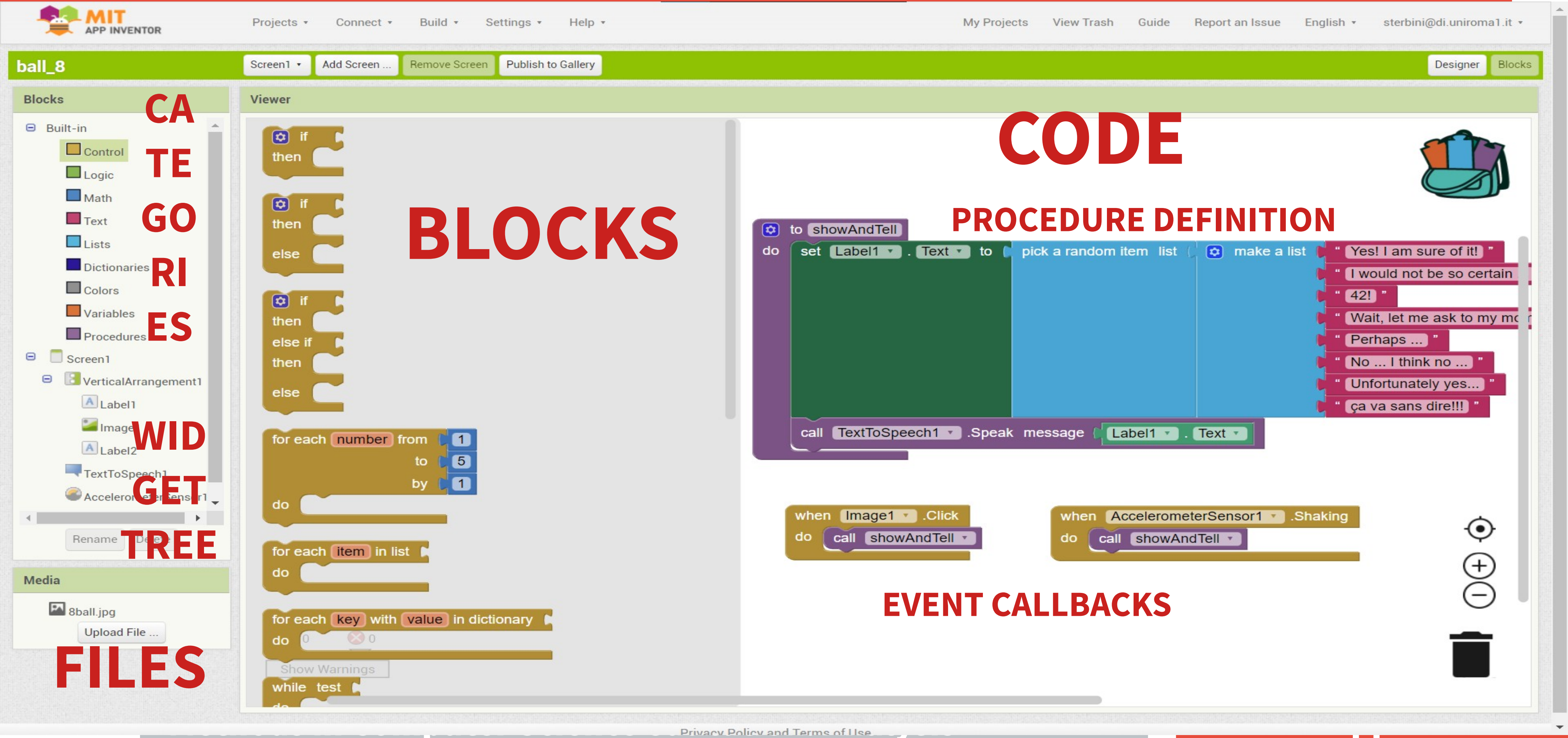
GUI EDITOR

WIDGET TREE

FILES

PROPERTIES

Code editor



CATEGORIES

BLOCKS

CODE

PROCEDURE DEFINITION

EVENT CALLBACKS

WIDGET TREE

FILES

```
to showAndTell
do  set Label1 . Text to  pick a random item list  make a list  " Yes! I am sure of it! "
                                                                " I would not be so certain
                                                                " 42! "
                                                                " Wait, let me ask to my mo
                                                                " Perhaps ... "
                                                                " No ... I think no ... "
                                                                " Unfortunately yes... "
                                                                " ça va sans dire!!! "
    call TextToSpeech1 .Speak message  Label1 . Text

when Image1 .Click
do  call showAndTell

when AccelerometerSensor1 .Shaking
do  call showAndTell
```

App structure

One “screen” for each phase (config, login, play levels, results …)

<u>Screens are independent</u> and DO NOT share data or code between them

(but you can use a local TinyDB key/value DB component that allows exchanging data)

Or you can pass/retrieve some text when switching to another screen

<u>Apps are independent</u> and DO NOT share data or code (Android)

(you can exchange data by using an external WebService + WebDB/CloudDB or with a Spreadsheet)

Resources (video, audio, files, images ...) are bundled in the apk

Practical Limit: <u>10 screens</u> max

To mimic many screens and share code between them you can hide/show widgets in the same screen by leveraging the widget tree (you just hide/show the parent widget)

Many widgets/objects available

Widgets:	Buttons and other input fields
Layout:	Automatic layout constraints (horizontal, vertical, grid ...)
Media:	Sound, Movie, Camera, SoundRecorder, SpeechRecognizer, TextToSpeech, YandexTranslate, …
Drawing:	Canvas, Sprite, Ball
Maps:	Maps, Polygonals, Markers, Features (from GeoJson)
Sensors:	Accel, Temp, Baro, Gyro, Barcode, Pedometer, NFC, …
Social:	Contacts, PhoneCall, Email, Twitter, Sharing, Texting
Storage:	TinyDB, TinyWebDB, CloudDB (Redis), File, DataFile (CSV/JSON), Spreadsheet
Connect:	BT Client, BT Server, Web, Serial, ActivityStarter (other apps)
Lego:	NXT, EV3

Data types

Numbers, Strings, Lists, Lists of Lists, Dictionaries, (Booleans)

All interface widgets are objects with:

- Predefined Properties (pre-set in the IDE, or read/changed by program)
- Events they can generate on interaction
- Methods that can be called

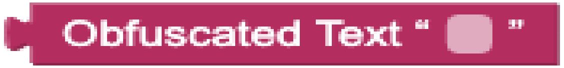
Some objects are not visual (i.e. BluetoothClient, File, DBFile, Sound, …)

Computed results are shown with a "puzzle" connector (while in Scratch they were ovals)

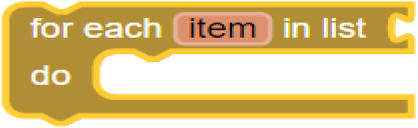
Some static data type enforcement is present (is checked but not shown)

NEW data types and methods

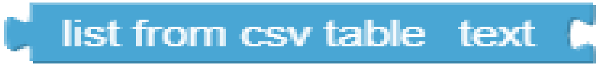
Text: obfuscated text


Obfuscated Text " "

Lists: foreach iterator


for each item in list
do

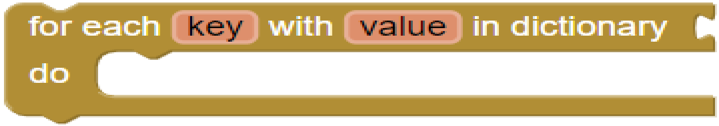
CSV <=> list of lists


list from csv table text

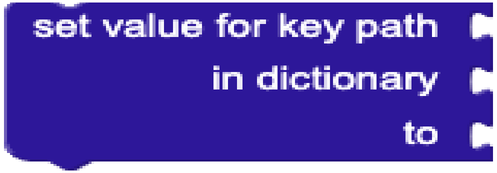
list of pairs as a read-only dictionary (FIRST match)

Dictionaries!

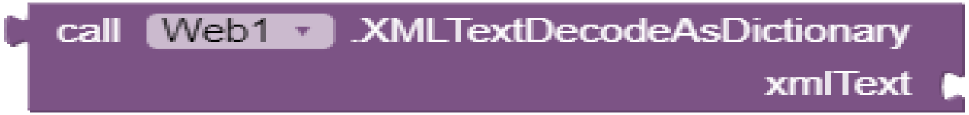
with key/value enumerator


for each key with value in dictionary
do

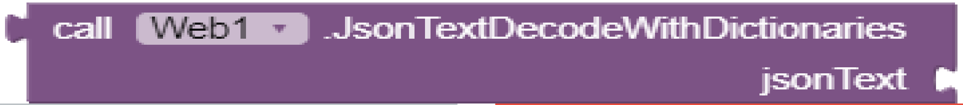
with path access to inner values


set value for key path
in dictionary
to

XML => convert to dictionary


call Web1 .XMLTextDecodeAsDictionary
xmlText

JSON => convert to dictionary

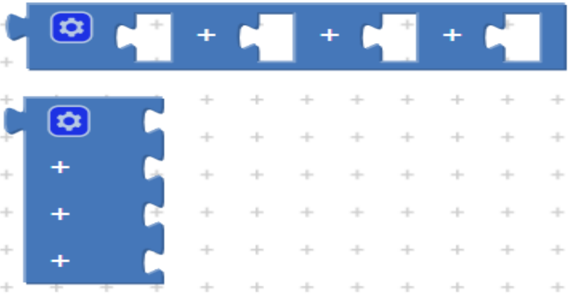

call Web1 .JsonTextDecodeWithDictionaries
jsonText

(Visual) Language style / Blocks symbology

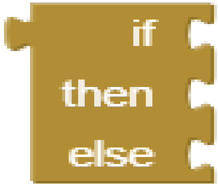
Inline or external inputs

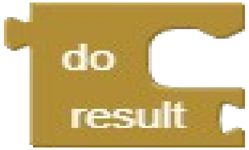
Extensible blocks to allow for more inputs

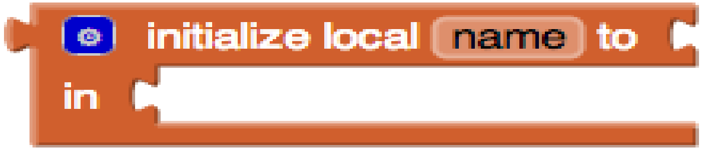
Text-based blocks (no pre-scholar)

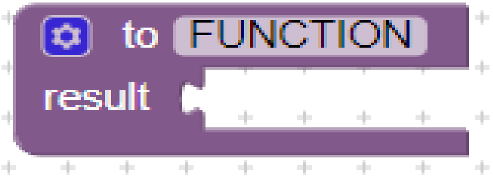


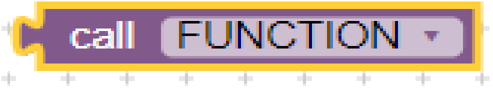
“Function-like” blocks (with result plug)



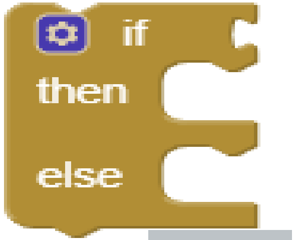


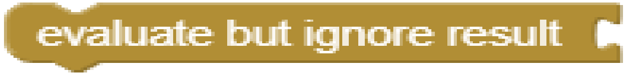


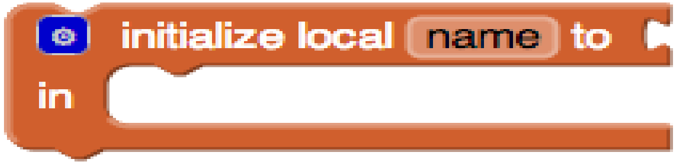


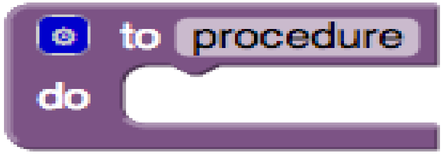


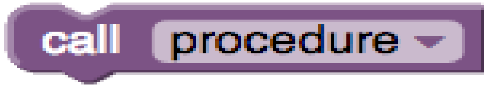
“Procedure-like” blocks (without result plug)











Code style: event-based

You implement mainly Events, Procedures and Functions

GLOBAL variables are defined outside any Event/Function/Procedure

You can define variables LOCAL to the procedure/function

- Can be changed/used only within their "scope bracket" (or as a return value)

This allows a "functional decomposition" style (but no lambdas/function passing)

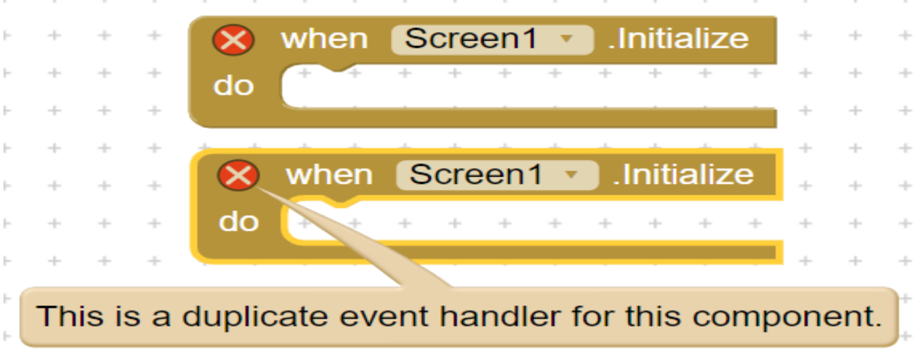
Limited support to debugging

- You can "collapse" the functions/events/procedures
- You can enable/disable some blocks
- Warning and Errors appear as yellow or red triangles
- All changes are automatically reflected in the Appinventor Companion app
- You can "Do it" a block and show the result
- You can "comment" your blocks

Execution model:
event-based programming

<u>NO multiple concurrent events</u>

<u>NO message passing</u>



This is a duplicate event handler for this component.

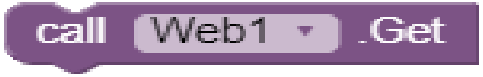
Almost all objects generate events when interacted with

E.g. “When the screen changes”, “When the button is clicked”,
“When got/lost focus”, “Before/After choosing an item”,
“When the screen orientation is changed”, “When the file has been read”
”When the web page has been retrieved”, “When the ball hits a border”,
”When the icon is dragged” . . .

Asynchronous protocols?

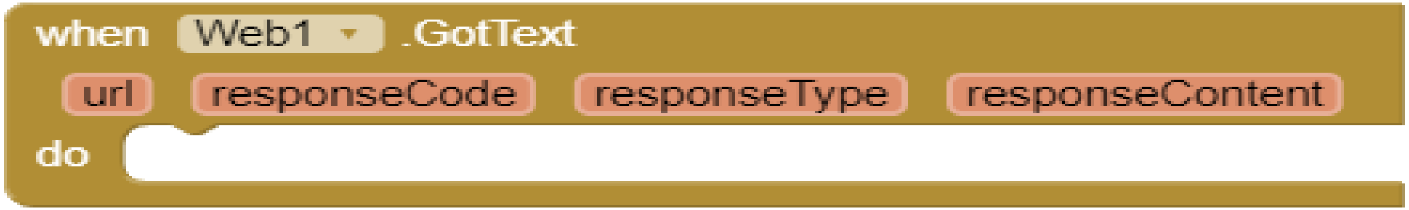
Asynchronous protocols are split in 2 or more phases

E.g. “Ajax query to web URL”



```
call Web1 .Get
```

“When the response arrives” events



```
when Web1 .GotText
  url  responseCode  responseType  responseContent
do
```

This to remove busy wait and to get an async interaction

To behave differently for different cases you can use globals as semaphores

PARTIAL object orientation (no way to add properties or to clone)

How to enable students’ cooperation

[Kate Feeney's MA thesis at the Mills College]

Ask each student to <u>implement just one screen</u> of a coordinated complex App

Start with a template App (just empty screens and media files)

Students should agree on <u>data interactions</u>, <u>data formats</u> and <u>names</u>

- Common resources can be shared among screens
- Communication between screens is handled by TinyDB objects

At the end you <u>merge all the screens made by the students</u> into a single App (with the AI2 Project Merger Tool)

Homework: build an app/game cooperatively

Other ways to organize collaboration projects

Multiple interacting applications can communicate through

- Bluetooth (direct communication + protocol implementation)
  (no async communication)

- Wifi + CloudDB (central coordination by data sharing)

Examples:

- Collect and map features on the field in real time (geolocalized data collection)

- Collect data from sensors and visualize them in real time (physics experiments)

- …

Extensions (written in Java/native)

ImageProcessor:	weighted combination of images
VectorArithmetic:	vector sum
SoundAnalysis:	pitch decoder (note recognition)
Posenet:	body pose estimation in a video (key joints and eyes/nose of a person)
BluetoothLE:	Bluetooth Low Energy
ScaleDetector:	pinch zoom/reduce
Look:	classify images/videos
ImageClassifier:	classify images/videos with your model

And MANY MANY MANY MORE!

Computational Thinking topics

Algorithm, structured coding, functions, local variables, data structures, types (enforced but not visually highlighted)

GUI programming, Event programming

NO simple concurrency (all events are single flow of computations + async)

More limited and easier than Snap! More powerful than Scratch

Mobile games

Multiplayer apps (connected by WebDB or Bluetooth)

Cooperative development!

Interdisciplinary topics ideas

So many sensors on a phone!!! →→ Physics experiments! Data collection!

Serial communication with Arduino →→ Home automation, robotics?

Protocol simulations with Bluetooth →→ Networks

NFC or QR codes →→ tangible interaction? Tagged info?

Maps, GPS, Maps Annotations →→ Geography, History, Geotagged data collection?

Media →→ Art, Literature

Text to Speech/Speech recognition →→ 2nd Language?

Lego EV3 →→ Robotics? Physics? ….

… please suggest!