MIT App Inventor 2

Andrea Sterbini – sterbini@di.uniroma1.it

CC BY NC SA

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)

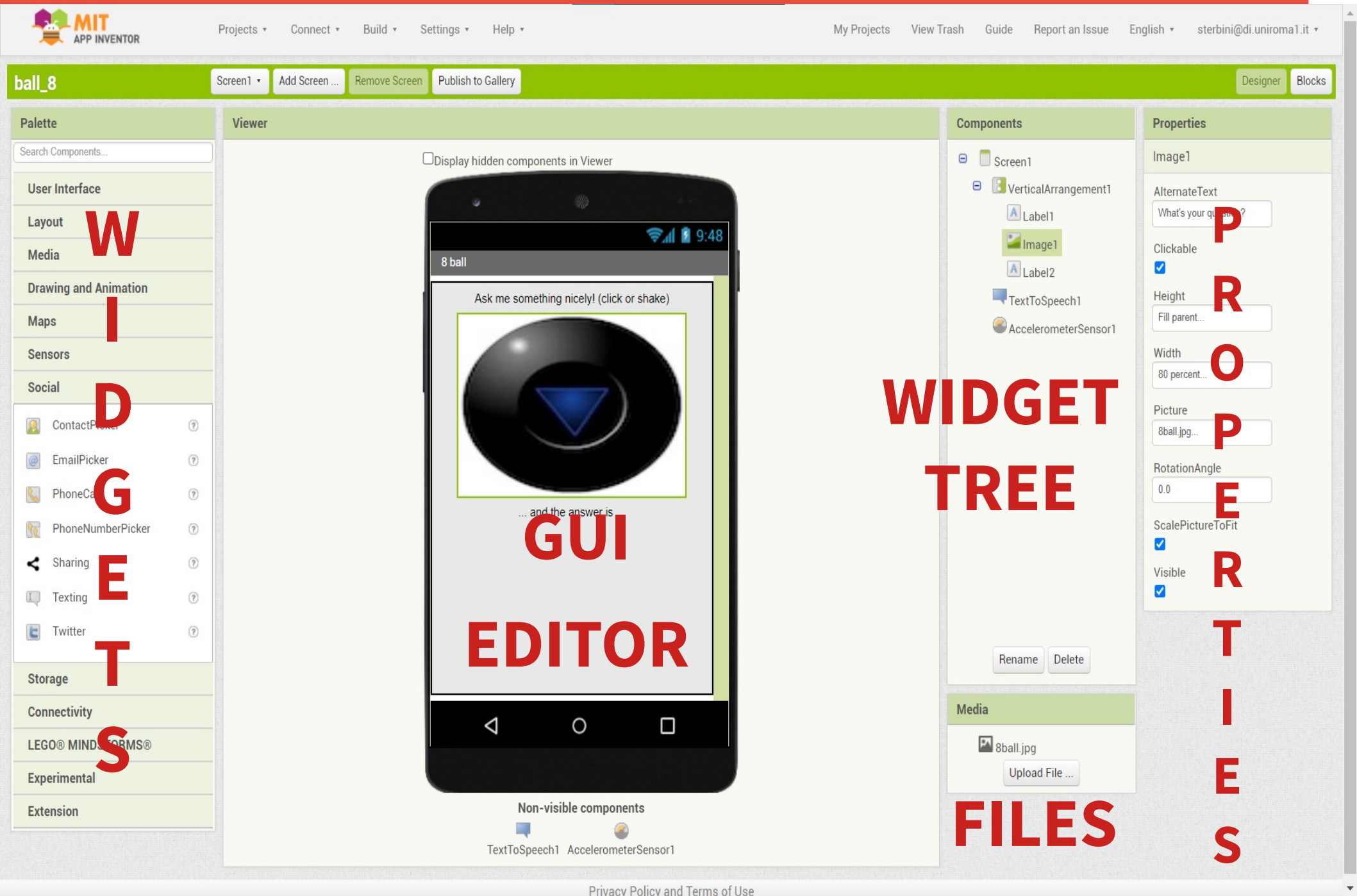
via USB or Wifi (if your phone allows it)

The server can be LOCAL to avoid network problems

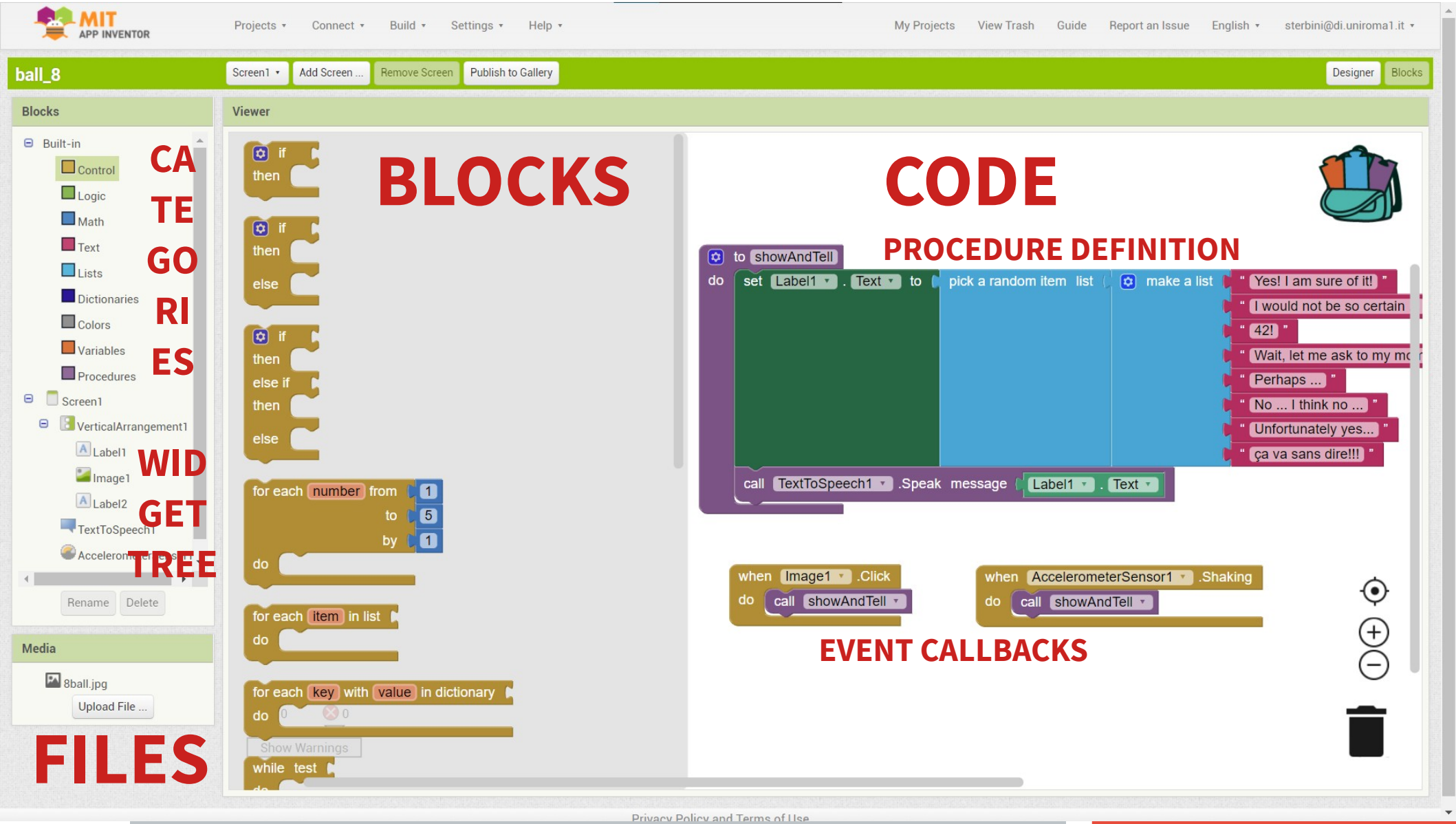
App Inventor 2 Ultimate [2018]

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

Web-based GUI editor



Code editor


MIT APP INVENTOR
Projects
Connect
Build
Settings
Help
My Projects
View Trash
Guide
Report an Issue
English
sterbini@di.uniroma1.it
ball_8
Screen1
Add Screen ...
Remove Screen
Publish to Gallery
Designer
Blocks
Blocks
Viewer
Built-in
Control
Logic
Math
Text
Lists
Dictionaries
Colors
Variables
Procedures
Screen1
VerticalArrangement1
Label1
Image1
Label2
TextToSpeech1
Rename
Delete
Media
8ball.jpg
Upload File ...
CATEGORIES
WIDGET TREE
FILES
BLOCKS
CODE
PROCEDURE DEFINITION
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
EVENT CALLBACKS
if then
if then else
if then else if then else
for each number from 1 to 5 by 1 do
for each item in list do
for each key with value in dictionary do
Show Warnings
while test
Privacy Policy and Terms of Use

App structure

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

Screens are independent and DO NOT share data or code

(but a local TinyDB key/value DB component allows exchanging data)

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

Apps are independent and DO NOT share data or code (Android)

(you can exchange data by using an external WebService + WebDB/CloudDB)

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

Practical Limit: 10 screens max

To mimic many screens 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

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, FirebaseDB

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, Sound, …)

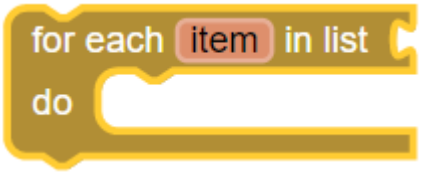
Computed values are represented with a "puzzle" connector (while in Scratch they were ovals)

Some static data type enforcement (checked but not shown)

NEW data types and methods

Text: obfuscated text

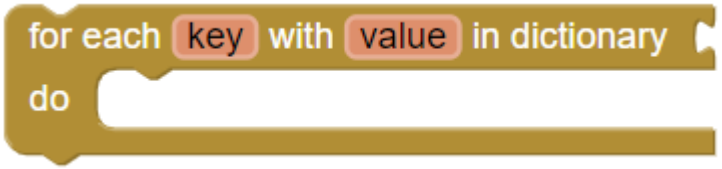
Lists: foreach iterator



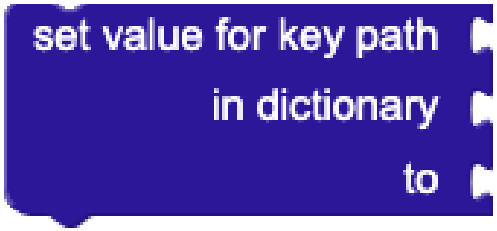
CSV <=> list of lists

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

Dictionaries!



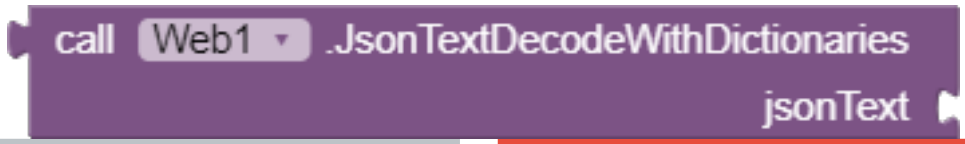
with key/value enumerator



with path access to inner values

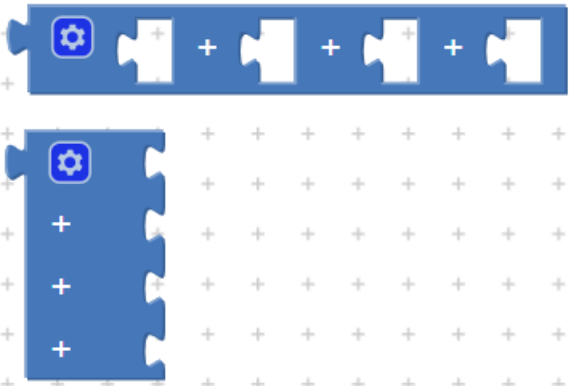
XML => dictionary

JSON => dictionary



(Visual) Language style / Blocks symbology

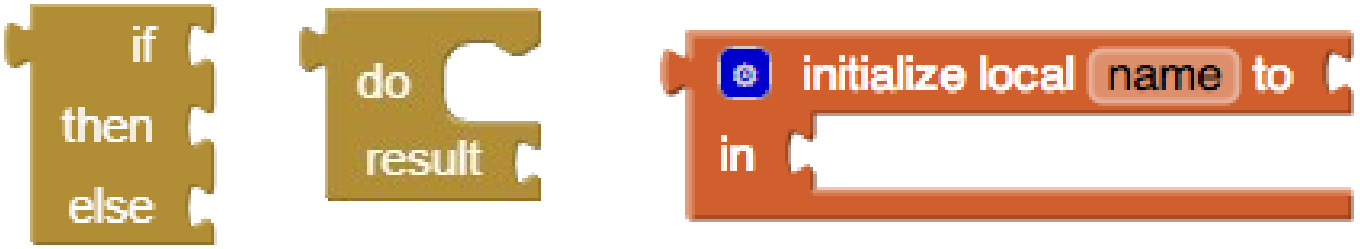
Inline / external inputs

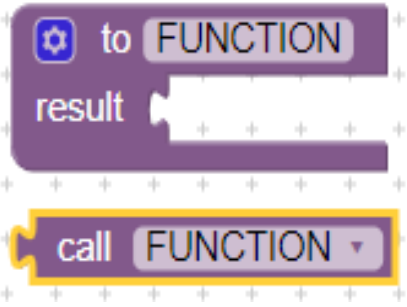


Extensible blocks

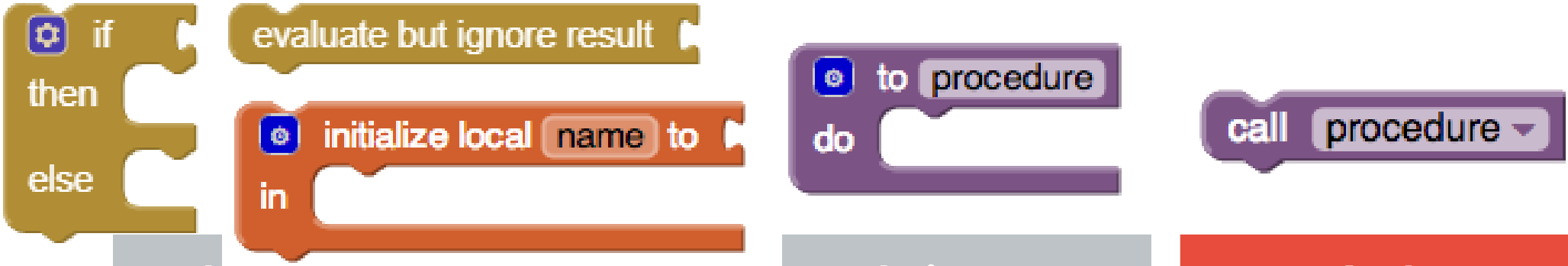
Text-based blocks (no pre-scholar)

“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 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)

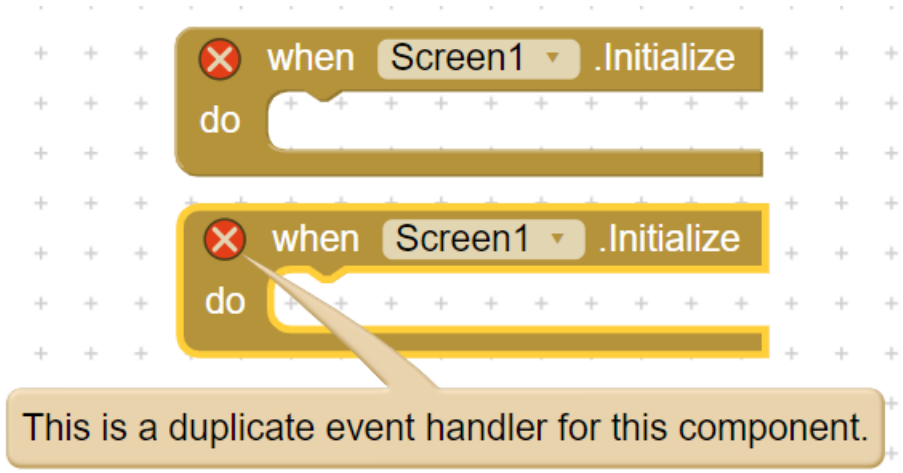
Limited support to debugging (e.g. NO easy variable display)

- You can "collapse" the functions/events/procedures
- You can enable/disable some blocks
- You can "comment" your blocks
- All changes are automatically reflected in the Appinventor Companion

Execution model: event-based programming

NO multiple concurrent events

NO message passing



Almost all objects generate events when interacted with

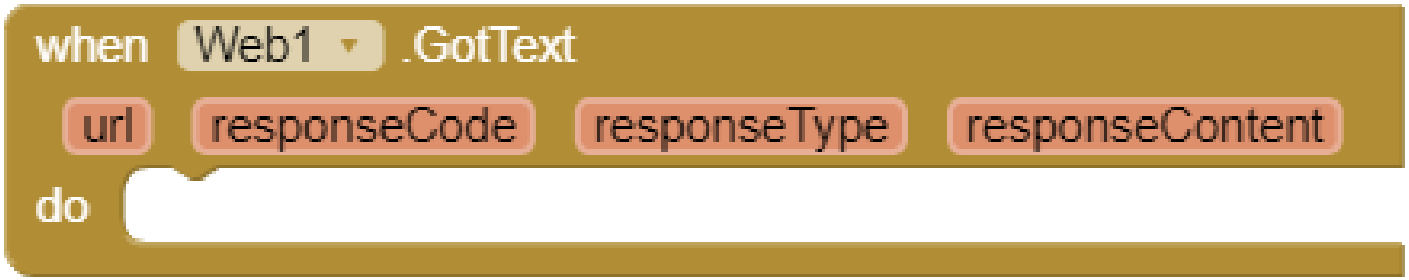
E.g. “When the screen changes”, “When the button is clicked”,
“When the text-area content is changed”, “When permission is granted”,
“When got/lost focus”, “Before/After choosing”,
“When the screen orientation is changed” …

Asynchronous protocols?

Asynchronous protocols are split in 2 or more phases

E.g. “Ajax query to web URL”

“When the response arrives” events



This to remove busy wait and to get an async interaction

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

NO 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 implement just one screen of a complex App

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

Students agree on data interactions, data formats and names

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

At the end you merge all the screens made by the students 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
- Collect data from sensors and visualize them in real time
- …

Extensions (written in Java/native)

ImageProcessor:	weighted combination of images
VectorArithmetic:	vector sum
SoundAnalysis:	pitch decoder (note recognition)
Posenet:	body pose estimation (key joints of a human)
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 (single flow of computation)

More limited and easier than Snap!

Mobile games

Multiplayer apps (connected by WebDB or Bluetooth)

Cooperative development!

Interdisciplinary topics ideas

So many sensors!!! →→ Physics! Data collection!

Protocol simulations with Bluetooth →→ Networks

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

Maps, GPS, Maps Annotations →→ Geography, History

Media →→ Art, Literature

Text to Speech/Speech recognition →→ 2nd Language

… please suggest!