# Drag and Swap Puzzles in JSON

Abe Getzler

Art and puzzle design by Red Scorpion

2/2017

[Drag and Swap Puzzles in JSON](#_aii17q83aghb)

[Motivation](#_wadke7eo8bis)

[The App](#_o3set4grg5pk)

[Designer](#_haanpqxor3yv)

[Other Components](#_t0wlp6m5ew2w)

[Media Drawer](#_gyq3wh69q0wi)

[Puzzle Storage](#_gq52bigqn9xk)

[puzzles.txt](#_1ixhqlshis1t)

[Puzzle1.json](#_na6hxac24wx6)

[Processing](#_txqx8wt7mno)

[Screen1.Initialize](#_g7ffzkjs3u0m)

[set\_sprite\_sizes](#_d7i84qkz5mgj)

[all\_sprites](#_h75it75oul45)

[When FilePuzzles.GotText](#_72c58ef24qt9)

[begin\_load](#_r2jjke3yb8a7)

[When File\_currentPuzzle.GotText](#_4fc20fu1z6mj)

[hint](#_rl400e2xo5ly)

[Puzzle](#_r9wlavuq7hl4)

[last\_save](#_dx8r2o53btl2)

[load\_sprite\_pictures](#_jmf9ju689mb)

[welcome](#_ah6bdnqwu3sf)

[Next and Back Buttons](#_nany7rfjlmz8)

[Puzzle mechanics](#_uo28sivf9c9n)

[When P1.Dragged](#_3yky9z6gw5o7)

[handle\_drag](#_n2ggcg3us9ot)

[handle\_touchDown](#_r7dpqnkug6p5)

[When P1.TouchUp](#_lvhufj13m5sl)

[handle\_TouchUp](#_m1u0tys7kp3a)

[is\_inside](#_3p84xoi9i1wa)

[swap\_pictures](#_2siymh7ec8l)

[save\_sprite\_pictures](#_73i8byvvn22k)

[snap\_back](#_yrjyrpms349l)

[btnHint.Click](#_cak6mn9b9oq3)

[completion\_level](#_7ar7jum3vp39)

[match\_count](#_dpqgdjom4ri9)

[end](#_u1wfa3j797k9)

[When CloseHint.Click](#_oqfwi546zwz3)

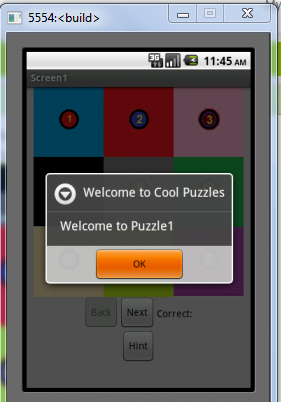
[Gallery Link](#_axcgv824u36v)

## Motivation

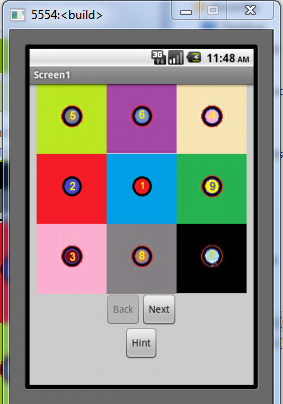
This sample app is a showcase for how to avoid sprite cannibalism in a drag and drop app, plus how to use JSON encoding to load complex data structures simply and flexibly. The evolution of the app can be seen in this AI2 board thread: <https://groups.google.com/d/msg/mitappinventortest/9kS4bONlhBY/8kBdoZ0nDAAJ>

Original puzzle design and art are by Red Scorpion.

## The App

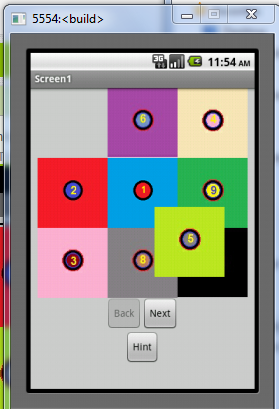


The app runs off different puzzle files, with a Welcome Notifier when a puzzle is loaded.

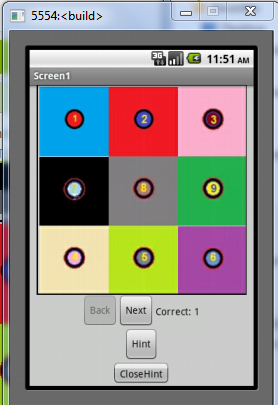


Back and Next buttons let the user go through a list of puzzles.

Each puzzle consists of an arrangement of 9 colored square sprites, that have to be dragged into the proper arrangement.



A square being dragged, mid-drag. When it is dropped, it will change places with the square it was dropped onto. Actually, it pops back to where it was before the drag, and swaps picture file names with the target sprite. (The result looks the same.)

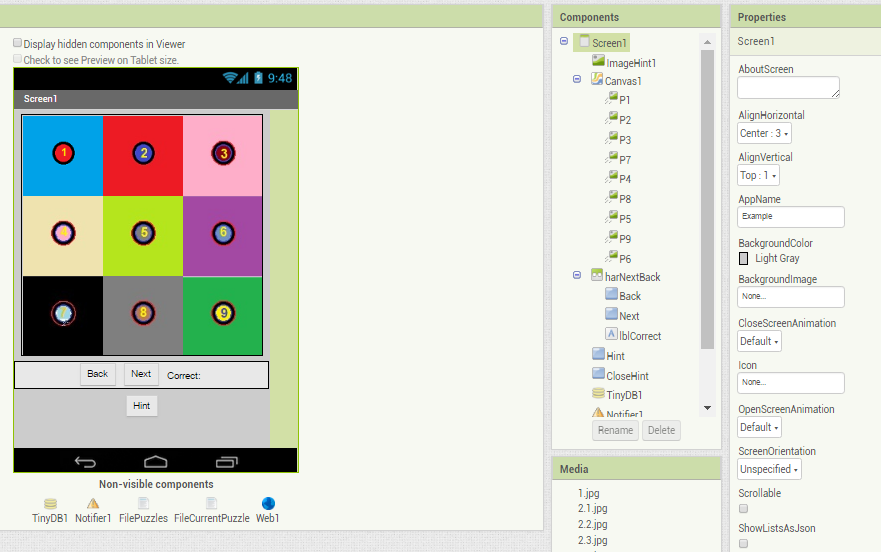


A hint button hides the Canvas and its Sprites, then reveals an image of that puzzle’s solution, and a label telling how many sprites have the right image in the right place.

A Close Hint button gets you back to the puzzle.

(Note for further work … the Hint buttons should also hide the Next and Back buttons, otherwise you could switch puzzles mid-hint.)

## Designer



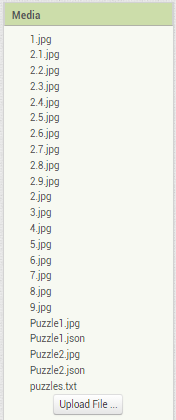
This app is all in Screen1, in fixed nonscrollable mode. The GUI consists of:

* ImageHint1, a 300 by 300 pixel hint image, normally hidden
* Canvas1, a 300 by 300 Canvas for the drag and drop puzzles,
  + Sprites P1-P9, 100 by 100 each, on the Canvas in a grid.
* A Horizontal Arrangement with:
  + btnBack and btnNext to decrement or increment the current puzzle number, and load the new puzzle number
  + lblCorrect, shown during the Hint display, to tell how many sprites are correct.
* Hint and Close Hint buttons. The Close Hint button only shows during hint display.

### Other Components

* TinyDB is used to save the current puzzle number, and each puzzle’s last arrangement of sprite images under the tag “PuzzleN/save” where PuzzleN is the puzzle name.
* Notifier1 is used for the Welcome popup, and can be used for other popups like Congratulations (not implemented.)
* FilePuzzles is a File component for loading the puzzles.txt file, which holds the names and file names of all the puzzles.
* FileCurrentPuzzle is a File component to handle loading a puzzle.
* Web1 is there for JSON conversion support, though no web access happens.

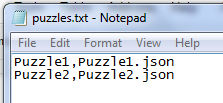
### Media Drawer



The Media Drawer has the uploaded puzzle sprite images, hint images, individual JSON puzzle files and a master file puzzles.txt pointing to the list of JSON puzzle files.

## Puzzle Storage

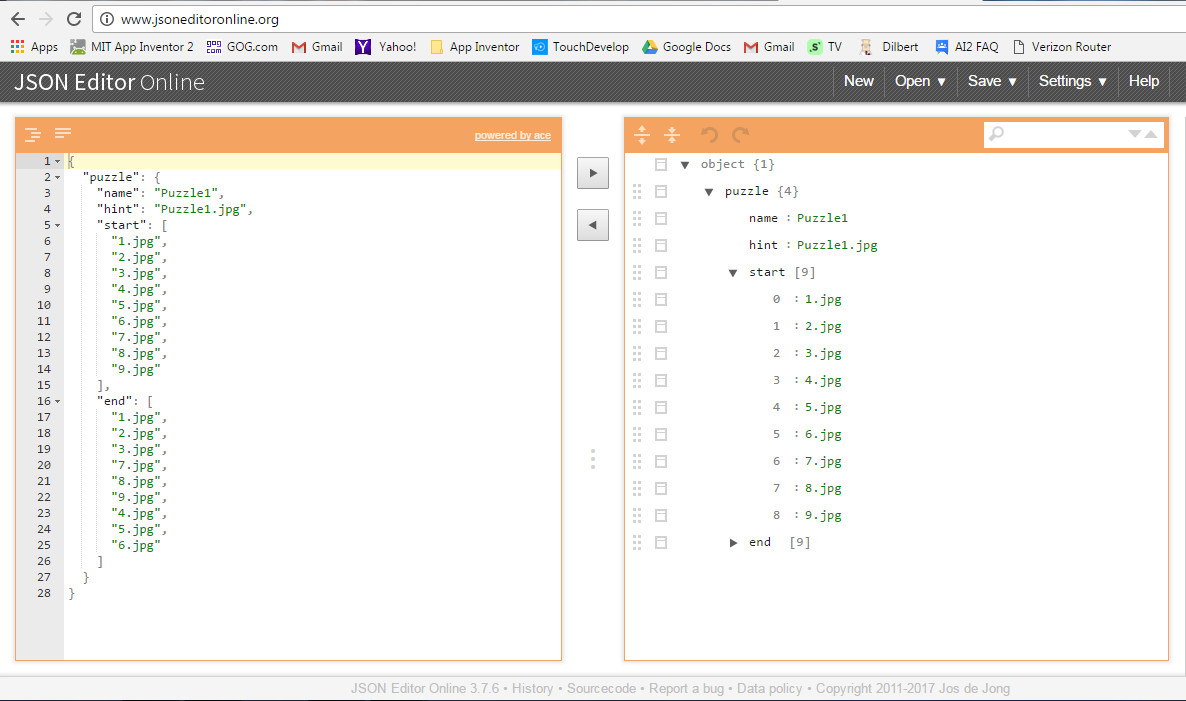
### puzzles.txt



This file is in Comma Separated Values (CSV) format, with one row per puzzle.

Each row has two columns, a puzzle name and the file name of the JSON file for that puzzle. The JSON file is read-only, with everything to start a puzzle.

### Puzzle1.json

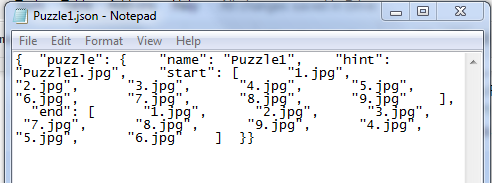


This is an image of the web site [www.jsoneditoronline.org](http://www.jsoneditoronline.org) where I found a very nice free online JSON editor. It has Load and Save facilities, and you can change the shape of your data, with no need to install any editor program.

(A Non-programmer’s introduction to JSON:

<http://blog.scottlowe.org/2013/11/08/a-non-programmers-introduction-to-json/>)

If you are offline, you can still use a simple program like Windows Notepad or Notepad++ to edit your JSON file, but you will have to be very careful not to break its structure.



The important lesson here is to not change any of the punctuation ({}:,[]) or the tag names (“puzzle”, “name”, “hint”, “start”, “end” ) used by this app to pick out pieces of data from the JSON file, otherwise that part of the app will fail on it.

The file being editted is Puzzle1.JSON, which describes Puzzle1, the first puzzle.

It has 4 parts:

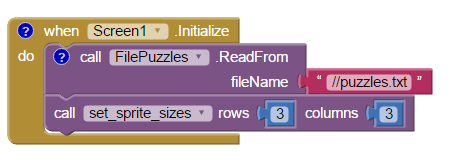
* **name** - the puzzle name
* **hint** - the name of the hint image file in the Media Drawer
* **start** - an array (list) of the starting picture file names for sprites P1 ...P9
* **end** - an array (list) of the target picture file names for sprites P1 ...P9 to judge completion (notice how the order is different from the start list, otherwise this would be a very boring puzzle.)

The four tag names mean nothing to JSON or AI2. They are wired into the blocks for this app, soon to come.

Given the sample files Puzzle1.json and Puzzle2.json, it should be possible to send a non-programmer away to develop more puzzles with an art program and a JSON editor, without changing any AI2 blocks. That’s a technique used by professional game development companies to save money, that you can use, now that you know JSON.

## Processing

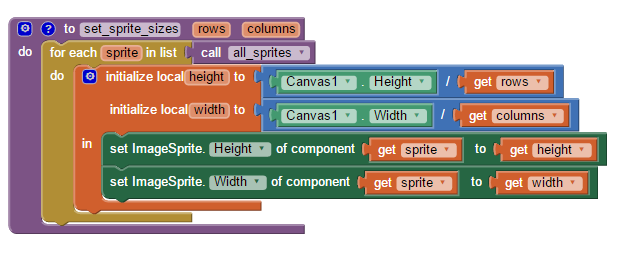
### Screen1.Initialize



Not much needs to be done at Screen1.Initialize, except launching a File Read of puzzles.txt from the Media Drawer (//) and adjusting the sizes of all the sprites to fit 3 by 3 puzzles.

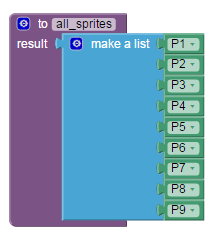
If this app is extended to include other shapes, like 4 by 4, the puzzle JSON files would need to include rows and columns attributes for each puzzle, and the [set\_sprite\_sizes](#ewa1oqyppevl) procedure would need to be called after each puzzle file is loaded, based on that puzzle’s rows and column values.

#### set\_sprite\_sizes



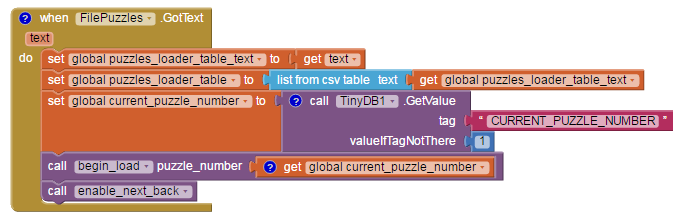
This procedure fits all the sprites returned by procedure [all\_sprites](#bmy45bpb5brz) onto Canvas1.

##### all\_sprites



This procedure returns a list of all the sprites in the game. It is a result procedure instead of a global variable because you can’t define component blocks before the app runs (note: AI2 has dropped this restriction - ABG) , and this way you don’t need to clutter up Screen1.Initialize with loading up the global variable (also no longer necessary) .

### When FilePuzzles.GotText

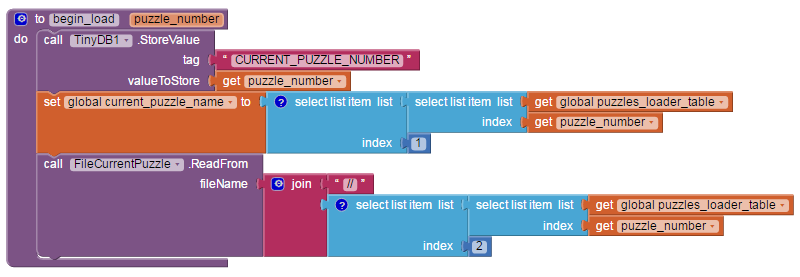


This event fires after the text from loading [puzzles.txt](#oer1ymmf0cet) arrives. For debugging purposes, a copy of the text is saved in a global variable **puzzles\_loader\_table\_text**, then run through a **list from csv table** block into the global variable **puzzles\_loader\_table**.

We check TinyDB tag “CURRENT\_PUZZLE\_NUMBER” to see if we were in the middle of a puzzle we should resume, otherwise start at puzzle 1, assigning it into global variable **current\_puzzle\_number**.

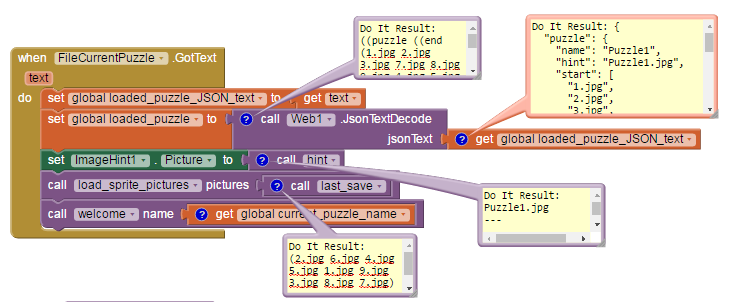
We call procedure [begin\_load](#1pl5rv1xcsuz) to start the loading process for the current puzzle number.

#### begin\_load



Each time we load a puzzle, that’s a good time to set the CURRENT\_PUZZLE\_NUMBER tag/value in TinyDB for the next run. SInce the global **puzzles\_loader\_table** has been loaded, we can refer to it by row and column, to set global current\_puzzle\_name and to start a read from the appropriate JSON puzzle file, named in column 2 of the [table](#oer1ymmf0cet).

### When File\_currentPuzzle.GotText



Following good debugging practice, we pass the incoming text through a global variable to allow us to see it in a Do It in the Blocks Editor. This is the same text we saw earlier for Puzzle.json before we transform it into a JSON list structure using the Web1.JsonTextDecode block and save it in the global **loaded\_puzzle** variable.

The hint image file name is now available, through the [hint](#1lenlrfaioak) function, so we load it into the ImageHint1 component Picture.

We now have access to the start sprite image list for this puzzle, so we can call [load\_sprite\_pictures](#f3dl54ifytv1) to change the sprite Picture attributes for all 9 sprites to either the [last save](#bbo856zalr1s) or the start values, if this is a new puzzle.

It’s also a good place to pop up a [welcome](#umqwzp5mw8h0).

#### hint

This procedure extracts the hint image file name from a puzzle JSON structure.It uses the [puzzle](#2fn1xmgihxhd) function to strip off the outer level to get at the pairs inside it using the list block **lookup in pairs**. The text block “hint” has to exactly match the tag “hint’ in the [JSON file](#gasq3r7lvg33) for this to work.

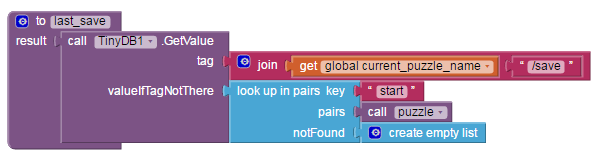


##### Puzzle



This procedure strips off the outer wrapper from the puzzle JSON tree and returns a list of pairs (attribute, value) with the four parts of a puzzle. The attribute names are the keys, not the order.

#### last\_save

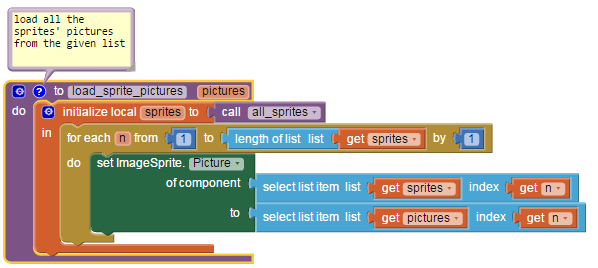


This procedure is intended to retrieve the last save of this puzzle, from Tinydb, if available. The save consists of the list of sprite Picture file names in the 9 sprites at the last save operation.

The TinyDB tag consists of a JOIN of the puzzle name (“Puzzle1”) and the extension “/save”. This lets us keep saves for all the puzzles.

If the TinyDB GetValue comes up empty, we look for the “start” tag in the current JSON [puzzle](#2fn1xmgihxhd) and use that.

#### load\_sprite\_pictures



This procedure receives a list of the sprite picture file names to be loaded into sprites P1, P2, … P9. Procedure [all\_sprites](#bmy45bpb5brz) returns our list of sprite components. We walk the list, index by index, and assign matching positions, using one of the Any Sprite component blocks.

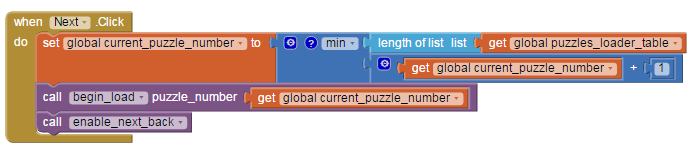
Bug warning … If this is extended to allow puzzles with less that the total number of sprites, say 2 by 2 puzzles, this would fall off the end of the list **pictures**. Using length of list(pictures) would probably fix that.

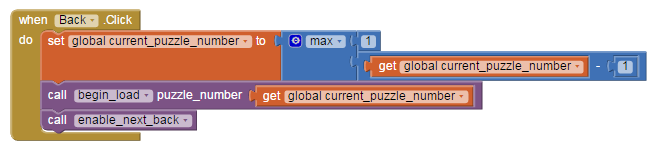
#### welcome



Not much to say here.

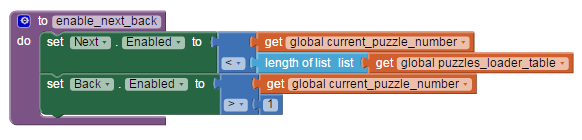
### Next and Back Buttons





The Next and Back buttons move the **current\_puzzle\_number** global variable forward or back, subject to the total number of puzzles in the global **puzzles\_loader\_table**.

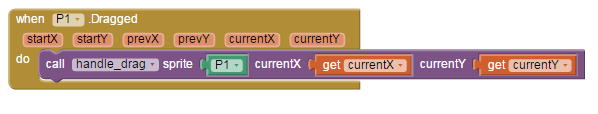
The Next and Back buttons are enabled or disabled depending on which limit has been reached.



This is double protection against falling off the end of the puzzles table.

### Puzzle mechanics

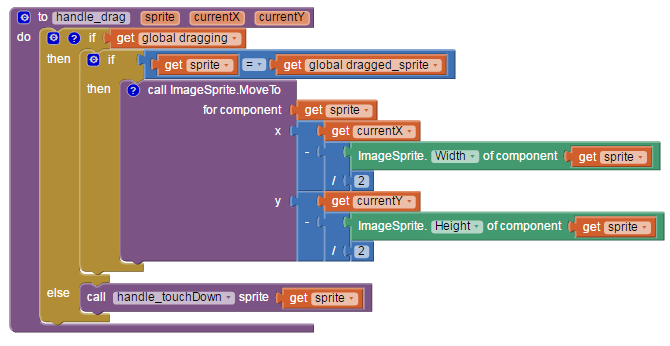
### When P1.Dragged



Each of the 9 sprites has a Dragged event block. They all call the common procedure [handle\_drag](#qs96jsiq35gx), passing it that Sprite component, and the currentX and currentY of the drag event.

(This was written before generic Events became available - ABG)

#### handle\_drag

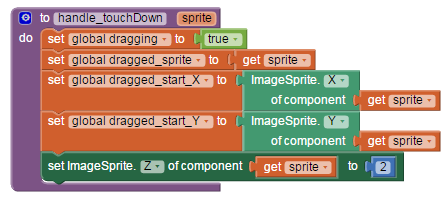


This procedure contains code to avoid sprite cannibalism, where extra sprites try to follow your finger as you drag another sprite near them. The two tests handle that - a global true/false value **dragging** to indicate that we are in the middle of a drag operation, and a global variable **dragged\_sprite** to hold the component that we decided to drag when we started a drag operation.

If this sprite is the currently dragged sprite, then we move it, with some adjustments so it doesn’t get obscured by our finger as it’s dragged.

If a drag operation starts while the dragging variable is false, then we handle it like it was a touchDown event, and call procedure [handle\_touchDown](#3wwifgprwzba).

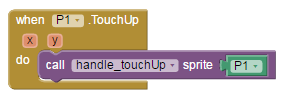
#### handle\_touchDown



This procedure is called at the start of a drag sequence. We announce in the global **dragging** variable that a drag has started, and save the current sprite parameter in the global **dragged\_sprite**. We also save the X and Y values of this sprite in the two globals **dragged\_start\_X** and **dragged\_start\_Y**, for the snapBack operation when we let go of the drag.

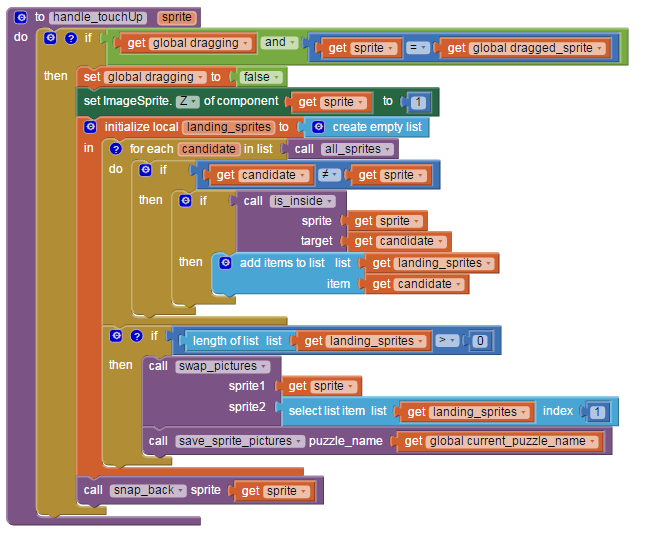
To show the dragged sprite better, we raise its Z value.

### When P1.TouchUp



The drag operation ends when we release the dragged sprite, signalling a TouchUp event for that sprite. We pass the component to procedure [handle\_touchUp](#6ggghqq0404i).

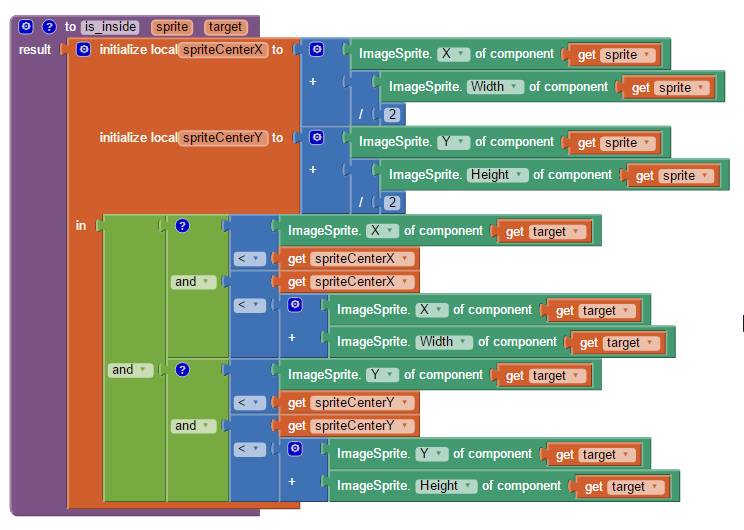
#### handle\_TouchUp



This procedure also protects against sprite cannibalism in the surrounding IF/THEN test, to insure it will happen only for the currently dragged sprite, and only during a drag operation.

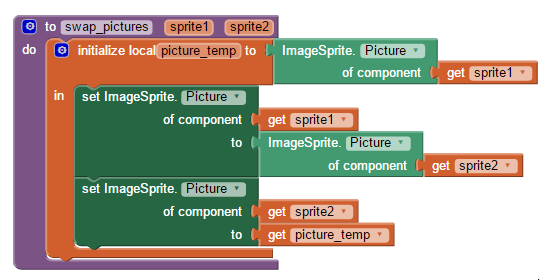
If the two tests are satisfied, it sets the global **dragging** variable false, and resets the Z of the sprite to 1. Since the drag might stop outside of any other sprite, or on the overlap of two sprites, we have to go through [all the other sprites](#_h75it75oul45) and check for overlap, using the procedure [is\_inside](#9b7a6mn740qz), collecting the result in local list **landing\_sprites**. If there are no landing sprites, there’s nothing to swap, and nothing to save. We take the first landing sprite in the result list to settle any tie, and call [swap\_pictures](#a56tv1v9p808) to swap pictures with that sprite, and [save\_sprite\_pictures](#1dknbbgvibfj) to save the new picture order in TinyDB for any future load operation. Either way, we snap the dragged sprite back to its original position using routine [snap\_back](#_yrjyrpms349l).

##### is\_inside



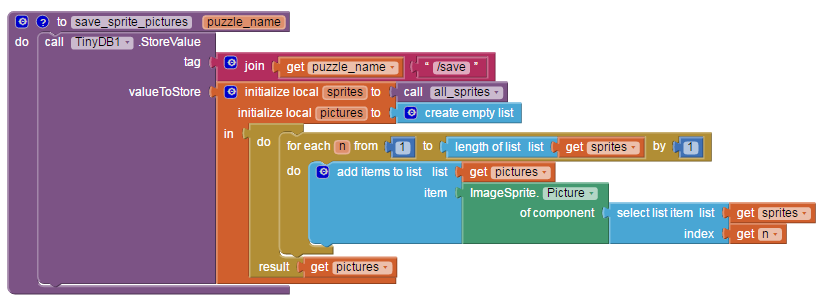
This procedure tests if a dragged sprite is inside a target sprite, judging by the center of the dragged sprite and the four walls of the target sprite. That comes out to 2 tests by X, and 2 tests by Y.

#### swap\_pictures



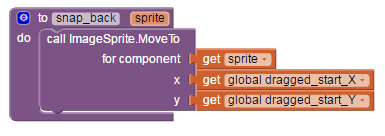
This procedure swaps the Pictures of two given sprites. It follows the common pattern of using a temporary variable to hold one of the swapped values, to avoid losing its value during the swap.

#### save\_sprite\_pictures



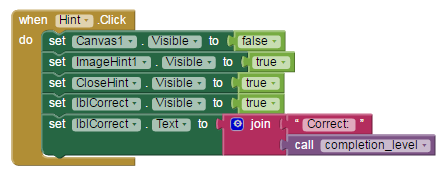
This works in the opposite direction of procedure [last\_save](#bbo856zalr1s), taking all the Picture file names from [all\_sprites](#bmy45bpb5brz) and building them into a list, then saving it in TinyDB under the PuzzleN/save tag for this puzzle.

#### snap\_back



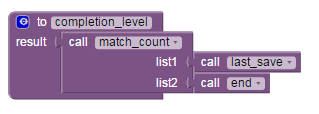
This procedure snaps the given sprite back to the X and Y position where the drag started, as save in the two global variables **dragged\_start\_X** and **dragged\_start\_Y**.

### When Hint.Click



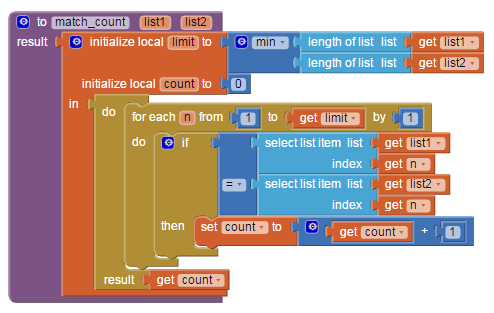
The Hint button hides the playing area and shows the hint image and the count of correctly placed sprite Pictures, as returned by procedure completion\_level.

#### completion\_level



The completion level of a puzzle is the count of how many of its sprite pictures ([last\_save](#bbo856zalr1s)) [match](#sssqfgpzylme) the [end](#jvrmaivy0qo7) value from the JSON puzzle file.

##### match\_count



This general purpose procedure will count how many items in one list match the corresponding position in another list. Extra care is taken to not run off the end of a short list.

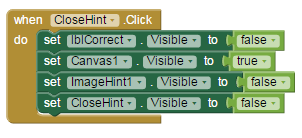
##### end



This procedure is similar to the [hint](#1lenlrfaioak) procedure, getting the end branch of the current puzzle JSON pairs returned by the [puzzle](#2fn1xmgihxhd) procedure.

This won’t work if the JSON file for this puzzle lacks the proper **end** tag and value.

### When CloseHint.Click



This button hides the hint elements from the GUI and shows the canvas.

## Gallery Link

[ai2.appinventor.mit.edu/?galleryId=6258216806383616](http://ai2.appinventor.mit.edu/?galleryId=6258216806383616)

## More Projects

<https://docs.google.com/document/d/1acg2M5KdunKjJgM3Rxpy_Rf6vT6OozxdIWglgbmzroA/edit?usp=sharing>