Kakuro Helper V01

```
Background
   Sample Puzzle
   Input Format
       Sample CSV Text
The Designer
   The Board GUI
          Label Attributes
          Button Attributes
Global Variables and Data Structures
       Button and Label Management Structures
              label table
              button table
       Game Loading Structures
       Game play structures
              game
              game history
          Cell ID calculation
              cell id
Processing
   Screen1.Initialize
       collect labels
       collect buttons
       build 2d board
       collect cell buttons
       cell buttons
       collect hars
       clear hars
       clear buttons
       clear labels
   Game Loading
       when File1.gotText
       collect Hsum cells
       collect Vsum cells
       Global range cells
```

```
Global range sums
      Collect sums
      Emit sums
display game
   display_button
   display label
      slash and label
          Z2
             blanks
Game Play
   when Button rc Clicked
   handle button
      global selected r c
          Get button
   When b1...b9 Clicked
      Handle digit
          Selected r
          Selected c
          Handle selection
             Replace cell
             Push
   Validate sums
      Cell id
      Collect parents
       Range is valid
          Collect children
          Sum of cells
          values of cells
             Game cell
             Row of cell
             Col of cell
          Duplicates
   Paint buttons
      Global invalid ranges
      Global invalid range cells
      Collect invalid cells
   Menu Buttons
      When btnBack Clicked
      Resume
```

When btnClear Clicked
When btnUndo Clicked
Pop
Display validated game
Validate all sums

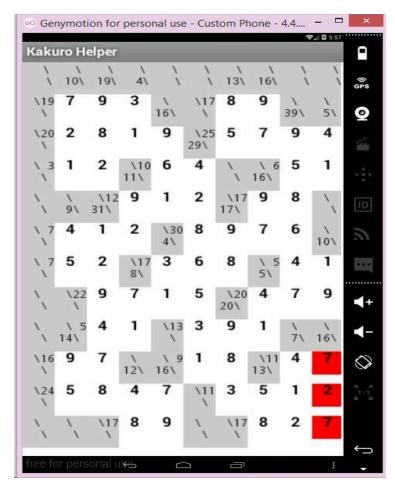
Gallery link
Other projects

Background

The New York Sunday News carries a Hidato Puzzle in its comic section. This app is meant to help solve the puzzle. This app lets you solve the puzzle, with Undo/Redo capability and error checking. Because of its size, special care needs to be taken in the Blocks Editor to keep everything collapsed, and to use a device or GenyMotion for testing. It's too big for the MIT Ai2 emulator.

Sample Puzzle

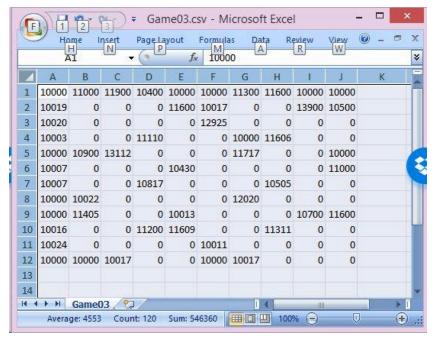
Kakuro is played on a 12 row by 10 column grid of black and white cells. The black cells contain target sums for the white cells immediately to their right and below them. The white cells start out empty, for the players to fill with numbers in the range 1 to 9. No duplicates are allowed in any vertical or horizontal range.



This sample puzzle has almost totally been solved, except for the vertical range of red cells in the lower right, which though it adds up to 16, contains duplicate 7's.

In the original puzzles, the black cells are drawn with white number sums on a black background, separated by a white diagonal line.

This implementation uses labels for the black cells, with black letters on a light grey background for readability. The diagonal line is simulated by two '\' characters, spaces, a line feed \n character, and a monospace font that just fits two lines per cell.



Input Format

Because of the grid nature of this game, it's easy to enter puzzles through a spreadsheet program. However, the black cells pose a problem: how to keep two numbers in one cell?

Because the white cells must contain unique numbers in the range 1 to 9, the sum of any range can't exceed 45 (1+2+...+9). If we express all range sums as two digit numbers, we can combine a vertical range sum VV and a

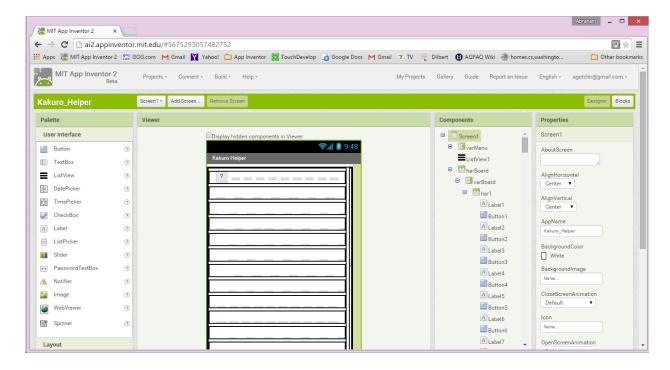
horizontal range sum HH as the number 1VVHH, like in this Game03.csv Excel file, which was used to load the preceding example. White cells start out with a value of 0. The leading '1' in the black cells is necessary to distinguish between a black cell with two blank range values (10000) and an empty white cell (0). This lets us use numeric input for all cells.

Sample CSV Text

This is how Game03.csv looks after being exported from a spreadsheet program in csv format...

```
10000,11000,11900,10400,10000,10000,11300,11600,10000,10000
10019,0,0,0,11600,10017,0,0,13900,10500
10020,0,0,0,0,12925,0,0,0,0
10003,0,0,11110,0,0,10000,11606,0,0
10000,10900,13112,0,0,0,11717,0,0,10000
10007,0,0,0,10430,0,0,0,0,11000
10007,0,0,10817,0,0,0,10505,0,0
10000,10022,0,0,0,0,12020,0,0,0
10000,11405,0,0,10013,0,0,0,10700,11600
10016,0,0,11200,11609,0,0,11311,0,0
10024,0,0,0,0,10011,0,0,0,0
10000,10000,100017,0,0,10000,10017,0,0,0
```

The Designer



The screen is composed of a few Vertical Alignments, only one of which is visible at a time. A Menu Vertical Arrangement is reserved for Menu operations, not yet implemented. An off-board ListView is used for button value choices.

The board is in its own Vertical Arrangement, visible when the app is built.

The Board GUI

The board uses Labels and Buttons for its cells. Other alternatives might have worked better, like a canvas with everything drawn on it, or a grid of List Pickers. I initially tried just buttons, but I was unable to align text in buttons to the right and bottom to get the diagonal line effect.

In order to support any configuration of labels (for the black cells) and buttons (for the white cells) in 12 rows of 10 cells, I used 12 Horizontal Arrangements stacked in a Vertical Arrangement. Each Horizontal Arrangement has 10 Labels and 10 Buttons, alternating. At game load time, each pair of label and button is visited, setting one of them visible and the other one invisible depending on cell type. That will support any sequence of labels or buttons

Label Attributes

These label attributes have to be specified in the Designer, because there is no Any block support for them...

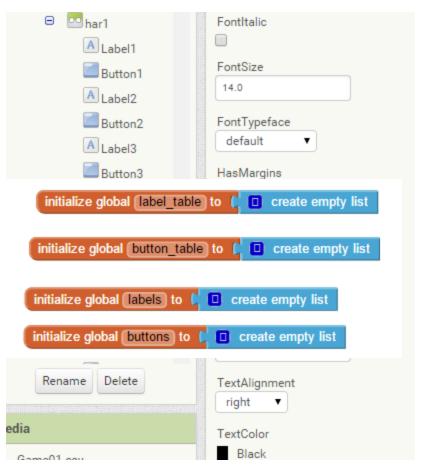
Labels must have no margins, and their Text Alignment must be Right. (Bottom would be nice, but it is unavailable.)

Label background color and text colors are set at game load time, along with height and width, which are set proportionally to screen height and width.

Button Attributes

Like labels, some button attributes must be set in the Designer because of lack of Block support...

Designer Text Alignment for buttons should be Center.



Global Variables and Data Structures

Button and Label
Management Structures
For convenience in the Blocks
Editor, all board buttons and
labels were loaded at
initialization time into the lists
buttons and labels.

Once the number of rows and columns was determined (currently forced to (12,10), the buttons and labels were gathered into rows and columns in button_table and label_table.

label_table

This contains the component

names of the board labels, assembled into a table (list of lists, row order).

button_table

This contains the component names of the board buttons, assembled into a table (list of lists, row order).

Game Loading Structures



A small number of pre-loaded game files were uploaded to the Media drawer, to be addressed via the game_files list. This is enough to get us

```
initialize global game_number to 5
initialize global ROWS to 12
```

started. A manifest file listing the game files would be more flexible.

Game play structures

game

This table contains the internal representation of the board. Labels are constant integers in the form 1VVHH where VV is a two digit vertical range sum of the cell range immediately below this cell, and HH is a two digit horizontal range sum of the cell range immediately to the right of this cell. The sums can range from 00 to 45. A zero sum can arise from the range being empty because of an adjacent label cell or from being on the edge of the board. The upper left cell must always contain 10000 because row 1 and column 1 contain only labels. The player cells start out with a value of 0 and can be filled with values 1 to 9 by the player as he fills in the board.

game_history

The Undo stack will be kept here. Copies of the global **game** table are taken after every move and stacked here in a list, making it easy to undo by unstacking a level and copying that level to **game**. (Not yet implemented.)

Cell ID calculation

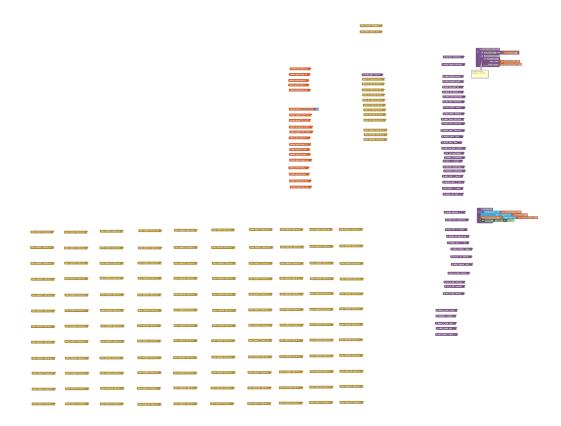
cell_id

```
to cell id
                                                  ialize global (game) to 📘 🔲 create empty list
result
                                     100
                            get 🔽
                                                   initialize global range_cells to 📜 🔲 create empty list
                            100
                                                   initialize global cell buttons to la create empty list
                                                                                make a list ()
                      (range id, sum)
                                                ? initialize global (range sums) to
                                                                                   create empty list
                                              initialize global (invalid_ranges) to
                                                                                create empty list
                                               initialize global invalid range cells to
                                                                                     create empty list
```

Each cell needs a way to identify it using a single value, combining both the row number RR and the column number CC of that cell. This function uses multiplication by 100 to calculate 1RRCC. The leading "1" is handy to force a double digit representation of both the row and column numbers, and to allow use of the text segment block to decode them.

Processing

Blocks Overview



All blocks have been minimized, and separated by function.

Note: This was written before the Al2 Generic Event block functionality was added. The grid of button Click events could collapse into a single Any Button Click event .

Screen1.Initialize

```
when Screen1 .Initialize
do set global labels to call collect_labels
    set global buttons to call collect_buttons
    set global label_table to call build_2d_board list get global labels rows get global ROWS cols get global COLS r
    set global button_table > to
                                call build_2d_board ist get global buttons rows get global ROWS cols get global COLS r
    set global cell_buttons > to (
                               call collect_cell_buttons button_table ( get global button_table v
    call clear_hars hars call collect_hars
    call clear_buttons buttons
                                get global buttons •
    call clear_labels labels
                             get global labels *
       initialize local gf to select list item list get global game_files index get global game_number
        call Notifier1 .ShowProgressDialog message ( Reading Game File )
        call File1 .ReadFrom fileName get gf
```

Because our labels and buttons will be addressed dynamically at run time, we need to collect them into lists at Screen1.Initialize time. The Horizontal Arrangements holding the buttons and labels are also collected in procedures. A few sample game files are available in the Media drawer, and one of them is loaded via the File1 component. (Game selection to be added later.)

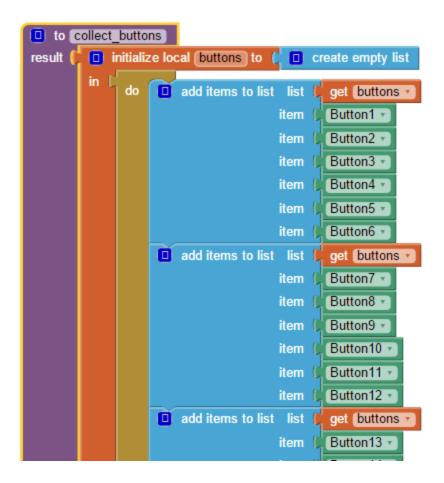
collect labels

```
to collect_labels
result
         initialize local (labels) to ( create empty list)
               do
                    add items to list list
                                             📜 get (labels
                                               Label1 ▼
                                        item
                                               Label2 •
                                        item
                                        item
                                               Label3 *
                                               Label4 •
                                        item
                                               Label5 •
                                        item
                                        item 🌘
                                               Label6 ▼
                    add items to list
                                        list
                                                get (labels
                                        item
                                               Label7 •
                                               Label8 •
                                        item
                                               Label9 •
                                        item
                                               Label10 →
                                        item
                                        item
                                               Label11 *
                                        item
                                                Label12 ▼
                    add items to list list
                                                get (labels
                                               Label13 -
                                        item
                                               Label14 •
```

To make all label components available to the Blocks Editor, they need to be put into some kind of list structure. I took the easy way out, and collected them in row-column order into a single list, 6 at a time because my laptop can't show big blocks. I pay the price later when I re-arrange them into a list of lists to match their row & column arrangement. This is a return procedure.

collect_buttons

To make all button components available to the Blocks Editor, they need to be put into some kind of list structure. I took the easy way out, and collected them in row-column order into a single list, 6 at a time because my laptop can't show big blocks. I pay the price later when I re-arrange them into a list of lists to match their row & column arrangement. This is a return procedure.



```
to build 2d board list
                            rows
                                    cols
result
        initialize local result to
                                     create empty list
           initialize local sublist to
                                     create empty list
                initialize local 11 to
        in
                   for each 🕝 from 🐞
                                              get rows 🔻
                                          create empty list
                        set sublist to
                        for each c from
                                                    get cols •
                             add items to list list
                                                       get sublist -
                                               item
                                                       select list item list 🍃 get (list 🔻
                                                                   index 🕻 get 🚺
                             set 🚺 to 🕻
                                               get 🔯
                        add items to list list get result
                                          item (
                                                  copy list list | get sublist v
                       get result -
              result (
```

For this version of the game, we have the number of rows and columns hard wired in two global variables. We reshape the one dimensional lists of labels and buttons into tables and store them into the global variables <u>label table</u> and <u>button table</u>.

collect_cell_buttons

cell_buttons

```
to collect cell buttons button table result in initialize local pairs to the length of list list the get button table by the do initialize local row to the select list item list the get result to the length of list list the get result to the length of list list the get result to the length of list list the get result to the length of list list the get result the list th
```

The global variable <u>cell_buttons</u> is initialized to a list of pairs, for use with the lookup in pairs block, to map a cell_id to its matching button component. It loops through the rows and columns of the global <u>button_table</u> and inserts a pair consisting of the <u>cell_id</u> of that row and column and that button component. This is a static structure.

collect_hars

```
result | make a list | har1 v | har2 v | har3 v | har4 v | har5 v | har6 v | har7 v | har8 v | har10 v | har11 v | har12 v |
```

The buttons and labels are arranged in rows in horizontal arrangements, which need to be treated alike, so I collect them all into a list, and feed the list to the routine that clears them all.

clear_hars

```
to clear hars
do
    for each har in list
                           get hars
          set HorizontalArrangement. AlignHorizontal v of component (
                                                                    get (har ▼
          set HorizontalArrangement. (AlignVertical v of component
                                                                  get (har 🔻
                                                                               to (
          initialize local h to
                                                                 get global ROWS
                                     Screen1 ▼
                                                  Height ▼
              set HorizontalArrangement. Height of component
                                                                  get (har -
                                                                              to
                                                                                   get [h ▼
          set HorizontalArrangement. Visible v of component
                                                                          to true
          set HorizontalArrangement. Width v of component
                                                                                           Width ▼
                                                             get (har •
                                                                         to 🛭
                                                                              Screen1 -
```

Each horizontal arrangement needs to pass down to its labels right-bottom alignment (3), and constant equal height and width.

clear_buttons

```
to clear_buttons
                    buttons
    for each (button) in list
                            get buttons *
         set Button. BackgroundColor of component get button of
         set Button. TextColor v of component (
         set Button. FontBold v of component (
                                                             to true •
                                              get button -
         set Button. Height of component
                                            get button 🔻
                                                                  Screen1 -
                                                                               Height ▼
                                                                                             get global ROWS
                                                          to
         set Button. Width of component
                                           get button 🔻
                                                                  Screen1 -
                                                                              Width ▼
                                                                                         🥊 get (global COLS 🔻
         set Button. Text of component get button of
                                                         to |
                                                             to 📗
                                                                  get global BIG_FONTSIZE
         set Button. FontSize v of component (
                                              get (button 🔻
```

To force all the cells with buttons on the board to a common size and appearance, we run through all the <u>buttons</u> and set their attributes. We will customize them further after we load a puzzle.

clear_labels

```
to clear labels
                  labels
    for each (label) in list
                          get (labels -
         set Label. BackgroundColor of component get label
         set Label. TextColor ▼ of component  get label ▼
         set Label. Height of component
                                          get [label ]
                                                                         Height / get global ROWS
                                                              Screen1 ▼
         set Label. Width v of component
                                         get [label ]
                                                             Screen1 ▼
                                                                         Width ▼
                                                                                  get global COLS -
         set Label. Text of component
                                        get [label ]
         set Label. FontSize v of component
                                           get (label 🔻
                                                         to get global TINY_FONTSIZE
         set Label. HasMargins v of component get label v
                                                           to false •
                                          get (label •
         set Label. Visible v of component
                                                       to false
```

To force all the cells with labels on the board to a common size and appearance, we run through all the <u>labels</u> and set their attributes. We will customize them further after we load a puzzle.

Game Loading

when File1.gotText

```
when File1 .GotText
 text
    call Notifier1 . DismissProgressDialog
                           list from csv table text
    set global game v to
    set global game history v to
                                   make a list
                                                    copy list list get global game
    set global range_cells v to ( call collect_Hsum_cells
    append to list list1
                                                  list2 call collect Vsum cells v
                         get global range cells
    set global range sums + to
                                  call collect sums v game
                                                              get global game
    call display game - Game
                                  get global game
```

The text in a game file is pure numerical csv table format, ready for a **list from csv table** block. We initialize <u>game</u> and <u>game_history</u>, collect range cells horizontally and vertically, calculate range sums, and display the game board.

collect_Hsum_cells

```
to collect_Hsum_cells
      initialize local cells to
             for each r from get global ROWS to 1 by 1
                get global COLS to (1) by (-1)
                 for each c from
                              call game_cell r
                                            get cells •
                                                     item C call cell_id r C get rv c G get c v
                                            get cells >  0
                              length of list list
                         initialize local (Hsum) to (
                                              ioin
                                                      · Ш ·
                                                           C call cell_id ▼ r
                         in for each item in list
                                            get cells •
                                add items to list list get result item (
                                                                  make a list 🕽 get (Hsum 🔻 🔰 get (item 🔻
                         result ( get result -
```

The result of this procedure will be a list of pairs (HRRCC, cell_id) where HRRCC identifies the row RR and column CC of the sum of the horizontal range that includes the cell identified by cell_id .Ranges usually have several cells in them, so there will be several pairs with the same HRRCC value but different consecutive cell_id values.

The ranges are identified by scanning game from right to left, building up an empty list of cells until a sum cell is hit, then posting the pairs for that range, clearing the list, and continuing up the row for the next range. Empty ranges are skipped.

0

collect_Vsum_cells

```
to collect_Vsum_cells
      initialize local (result) to
                               create empty list
          initialize local cells to 👂
                               create empty list
                               get global COLS to (1) by (-1)
               get global ROWS v to 1 by -1
                    for each n from (
                                                                           < 10000
                                   call (game_cell ▼ r
                                                                     call cell_id r | get rv c | get cv
                                   length of list list
                                                                        call cell_id r
                                 for each item in list
                                                     get cells 🔻
                                                           get result v item
                                                                              make a list
                             set cells to
                   get result -
```

The result of this procedure will be a list of pairs (VRRCC, cell_id) where VRRCC identifies the row RR and column CC of the sum of the vertical range that includes the cell identified by cell_id. Ranges usually have several cells in them, so there will be several pairs with the same VRRCC value but different consecutive cell_id values.

The ranges are identified by scanning <u>game</u> from bottom to top, building up an empty list of cells until a sum cell is hit, then posting the pairs for that range, clearing the list, and continuing up the column for the next range. Empty ranges are skipped.

Global range_cells

The horizontal and vertical range-cell pairs are collected into a single global list of pairs, range_cells.

Global range_sums

This global list is a 2 column table mapping range names into the target sums. We build it at game load time from procedure collect sums.

Collect sums

```
o to collect_sums game
        initialize local sums to 🕻 🧿 create empty list
         initialize local contents to (
                  for each r from 11 to
                                            get global ROWS by (1
                      for each c from 11
                                                 get global COLS •
                      do set contents ▼ to
                                               call [game_cell •] r (
                                        get contents -
                                                       > 10000
                                  append to list list1
                                                     get sums 🔻
                                                     call [emit_sums *
                                                                        call cell_id ▼ r 🔰
                                                                                        get rv c get cv
                                                             contents
                                                                        get contents 🔻
                      get sums
```

This procedure traverses a given game table, by row and column, checking each game cell if its **contents** are of the form 1HHVV, adding to a return list **sums** the extracted values from procedure **emit sums**.

Emit_sums

```
o to emit_sums (id) contents
                             remainder of 🔻
                                           quotient of 🔻 🛭
                                                                                 ÷ ( 100
                                                                         100
                                                        get contents •
                             remainder of v get contents v
                             create empty list
        initialize local sums to
                o if
                           get VV > 0
                                                             make a list io join ("V" get id
                                           get sums item
                o if
                           get HH > > 0
                                           get sums *
                                                             make a list
                                                                           join (
                                                                                   " (H) " | get (id )
            result | get sums •
```

These range sums are going to be needed for later, when the user works the puzzle. The VV (vertical) and HH (horizontal) sum parts of the given range cell are extracted, and are identified by a range type letter V or H prefix to the range cell ID, paired with the sum value. The (id,sum) values are collected in pairs, suitable for use with the lookup in pairs block.

display_game

```
to display_game Game

do

for each r from 1 to length of list list get Game v index get r

in for each c from 1 to length of list list get row by 1

do initialize local cell to select list item list get row index get c in call display_button value get cell r get r c get c button_table get global button_table call display_label value get cell r get r c get c label_table get global label_table r
```

To display the <u>game</u> table, we scan each row r and column c, and pass that cell, button, and label to routines display_button and display_label, which will decide to display or hide their respective component based on the cell value.

display_button

```
to display_button
                  value
                                   button_table
initialize local (buttons_row) to
                                    select list item list | get | button table |
                                                                            index 🌗
     initialize local button to
                                  select list item list get buttons row
                                                                           index (
                                                                                   get C
         if
                                   < 7 ( 10000
                       get value -
         then
                set Button. Text v of component get button v
                                                                 to get value •
                set Button. Visible v of component (
                                                    get button -
                                                                   to true •
                set Button. Enabled of component
                                                                    to true
                                                     get button
         else
                set Button. Visible v of component (
                                                    get button -
                                                                        false •
                set Button. Enabled of component
                                                                    to false
                                                    get button
```

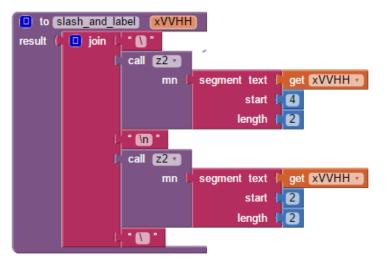
To display a button, we start by looking up its component in the <u>button_table</u>, by row and column, verifying that it is indeed a button by value < 10000, then set the button contents, visibility, and enabled attributes accordingly.

display_label

```
to display_label value
                           r c
    initialize local labels_row to
                                    select list item list | get (label_table
        initialize local (label) to
                                   select list item list | get [labels_row |
                                                                         index
            if
                         get value → 2 → 10000
                   set Label. Text of component get label
                                                               to call slash_and_label xVVHH get value x
                   set Label. Visible of component
                                                    get (label 🔻
                                                                 to [ true *
                   set Label. Visible of component
                                                    get [label •
                                                                 to false
```

To display a label, we start by looking up its component in the label_table, by row and column, verifying that it is indeed a label by value >= 10000, then set the label contents, visibility, and enabled attributes accordingly. The diagonal slash separating the vertical and horizontal sums is built up in the slash_and_label value procedure, and requires a monospace font and right-bottom alignment.

slash and label



This routine will take a 5 digit number 1VVHH and show it as

\VV HH\

The routine <u>Z2</u> handles forcing a constant width of 2 places, with leading blanks.

```
1 (2) to [22] [mn]
        initialize local m to
result
                                  segment text
                                                 get mn 🔻
            initialize local n to
                                  segment text
                                                 get mn •
                                                                        length |
                     compare texts
                                      get mn v
              then
                     call blanks w
              else
                                             get m 🔻
                             compare texts
                     then
                                       call (blanks v w
                      else
                             get mn
```

This routine replaces all leading blanks of a two digit number, replacing them with a blank. Because the Block Editor seems to eat blanks from text blocks, I had to write a value function that would return a requested number of blanks ...

blanks

```
result in initialize local lotsablanks to in segment text get lotsablanks very start length get w.
```

Game Play

To play the game, the user pushes one of the buttons on the board, then selects a ListView selection for what digit he wants to fill in for that cell, (or some menu options, to be added later.)

when_Button_rc_Clicked

```
when Button17 v. Click do call handle_button v

when Button27 v. Click do call handle_button v

when Button27 v. Click do call handle_button v

call handle_button v

do call handle_button v

r 13

c 7
```

handle_button

global selected r c

```
to handle_button r c

do set global selected_r c v to make a list get r v get c v

set harDigits v . Visible v to (true v

set Button. BackgroundColor v

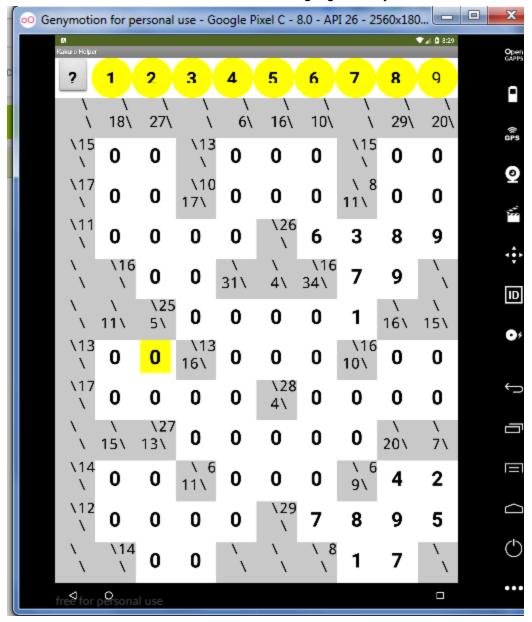
of component (call get_button v

c get c v
```

Each Button Click handler calls <u>handle_button</u>, which saves the selected row and column number in <u>global selected_r_c</u> as a pair, for later use after ListView1 returns a Selection. The button at (r,c) is looked up by routine <u>get_button</u>, then is colored yellow, so the player can choose a digit for that cell.

There are lots of alternative ways to have handled this, like a table of active and inactive text boxes.

The row and column of the selected cell are highlighted in yellow:



Notice how the input digits are exposed at the top, and also colored yellow.

Get_button

```
result ( select list item list ( select list item list ) get global button_table v index ( get c v
```

This is a 2 dimensional lookup into table button_table, initialized earlier.

When b1...b9 Clicked

Digit entries are handled by the handle digit procedure, taking as input the requested value to be assigned at the highlighted cell as recorded in the global selected_r_c variable. Because I needed to fit a Menu button onto the board at the left of digits, 1-9, clearing a cell to 0 is handled under the Menu button by the btnClear button.

Handle_digit

```
to handle_digit n

do initialize local r to call selected_r r

initialize local c to call selected_c r

in call handle_selection r

r get r r

c get c r

selection get n r
```

The first level of functionality for setting a cell value, identifying the row and column, is done by procedures <u>selected r</u> and <u>selected c</u>. Then we call <u>handle selection</u> with the row, column, and desired value that was just selected. Afterwards, we hide the Horizontal Arrangement with the input digits.

Selected_r

```
to selected_r
result select list item list get global selected_r_c index 1
```

Selected_c

```
to selected_c

result select list item list get global selected_r_c index 2
```

Handle_selection

```
to handle selection
                                 selection
do
    initialize local (button) to
                                 call get_button r | get r c | get c r
         call replace cell *
    in
                      table
                              get global game 1
                              get 🔽
                              get C
                              get selection *
                     value
                                                     get global undos •
         call push r item
                             get global game 🔻
                                                to
         set global redos T to 🚺 🔲 create empty list
         set btnRedo . Enabled to
                                         false 🔻
         call validate sums ▼ r 🕽 get r ▼
                                                get C 7
         set Button. Text of component
                                           get button -
                                                         to
                                                             get selection •
         call paint buttons
                button table
                                get [global button_table *
                               get global invalid ranges
               invalid_ranges
```

To handle selection of a value for a cell, we first have to identify which button is associated with that (r,c) coordinates, using procedure get_button, and save it in local variable button.

We then call procedure replace_cell to update our game board, then we push the new copy of the game board onto our global undos stack. A new move clears the global redos stack.

We disable the Redo button (hidden in the Menu) until we have something to redo.

We call procedure validate_sums to collect the ranges that need to be highlighted because they are clearly in error. The selected button has its .Text set to the requested selection value.

We then call paint_buttons to refresh each button with its validity indicator, white (valid) or red (invalid) background.

Replace_cell

```
to replace_cell table r c value

do initialize local row to select list item list get table r index get row replacement get value r
```

Cell value replacement is done in place in a two dimensional table (list of lists).

Push

```
to push item to
do add items to list list get to recopy list list get item
```

Pushing and popping are set up to use the last (highest index value) item of the stack as the most recent.

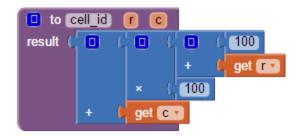
Validate_sums

```
to validate sums
do
     initialize local cell to
                                  call cell id r | get r v
     initialize local cell ids to
                                   create empty list
                                     create empty list
     initialize local parents to
     in
          set parents • to
                               call collect_parents ▼ child cell
          for each range in list
                                    get parents
          do
               Θ
                            call range is valid ▼ r
                                                       get range
                       call remove *
               then
                                   list
                                          get [global invalid_ranges]
                                value
                                          get range 🔻
                          add items to list
                                                   get global invalid ranges
               else
                                           item
                                                   get range *
```

Validation of ranges is done incrementally, just for the two ranges (horizontal and vertical) that contain the last updated cell. A global list of invalid ranges is maintained for the game board, for later use painting invalid cell ranges. This is a bit shaky, done for speed, to avoid duplicate work. If that global invalid_ranges list gets out of sync with the board, error painting will be thrown off.

We identify the **cell** using a single value, returned from procedure **cell_id** from its coordinates **r,c**. We call **collect_parents** to gather the parent ranges of our **cell** into local list **parents**. For each **range** in the list of **parents** we test if **range_is_valid**. If valid, we **remove** it from the global list invalid_ranges, otherwise we add it to that list.

Cell id



The id of a cell is a number of the form 1RRCC where RR is the row r, and CC is the column \mathbf{c} , as 2 digit numbers. This gives us a single value, for easy comparison.

Collect_parents

```
to collect_parents
                      child_cell
        initialize local owners to create empty list
result
                   for each (item) in list ( get global range_cells •
                        o if
                                           get child_cell *
                                           select list item list
                                                                            index ( 2
                                                              get item •
                               o add items to list list
                        then
                                                         get owners •
                                                          select list item list 🔰 get (item 🕶
                                                                                          index [1]
               result
                        get owners
```

This procedure references global <u>range_cells</u>, a two column table (range,cell) which was built during <u>game loading</u>. It builds up an initially empty list of **owners** from column 1 if column 2 matches the given cell id.

Range is valid

```
to range_is_valid
result
        o initialize local cells to
                                    call collect_children range
                                                                  get 🔽
                                      call sum_of_cells v cells
                                                                 get cells •
                                     look up in pairs key 🌹 get 📧
                                                           get global range_sums
                                               notFound
                                                          0
                       or 🔻
                                      call min of cells r cells
                                                                get cells
                              0
              and 🔻
                         0 = 7
                                    call duplicates . I
                                                         call values of cells cells
                                                                                      get cells •
```

For the game Kakuro, there are 2 rules that are applied to ranges that have been filled in (no zeroes):

- 1. No duplicates allowed
- 2. The sum must match the stated target value for that range.

First we <u>collect the children</u> of the given range r into local variable **cells**, a list of their cell ids. We count the number of <u>duplicates</u> of the <u>values of those cells</u>. There should be 0 duplicates. Then we check if we have a full range (**min_of_cells** > 0) and if the **sum_of_cells** of that range matches the value for that range in global range_sums.

Collect_children

```
to collect_children
        initialize local children to
                                         create empty list
result
                    for each item in list
              do
                                           get global range_cells •
                         🔞 if
                                            get range
                                    Œ٧
                                            select list item list 🕻 get (item 🔻
                                                                             index [1]
                                add items to list list
                                                           get children
                                                   item
                                                           select list item list
                                                                             get item
                                                                                            index ( 2
                        get children
```

This procedure traverses the global <u>range_cells</u>, returning a list of all the cells that are in the given range.

Sum_of_cells

```
to sum_of_cells cells
result in initialize local sum to initialize local sum t
```

This procedure does a running subtotal of the values returned from game cell at the row of cell and col of cell for the given cells list.

values _of_cells

```
to values_of_cells cells
result ( in initialize local values to ( create empty list in do for each cell in list ( get cells > do  add items to list list ( get values > item ( call game_cell > call row_of_cell > cell ( get cell > call col_of_cell > cell )
```

To collect the **values** of a list of **cells**, we look up each <u>game_cell_value</u> for the <u>row_of_cell_and</u> col_of_cell of that cell and add it to our return list of **values**.

Game_cell

```
to game_cell r c

result select list item list select list item list get global game index get r

index get c
```

Row_of_cell

```
result remainder of quotient of get cell ÷ 100 ÷ 100
```

Cell IDs are of the form 1RRCC, so this extracts the RR.

Col_of_cell

```
to col_of_cell cell

result remainder of get cell ÷ (100)
```

Cell IDs are of the form 1RRCC, so this extracts the CC.

Duplicates

```
o to duplicates
         o initialize local values to
                                        o create empty list
result
                  initialize local n to
                    for each item in list
                                            get [
               do
                    do
                          0
                                       get item *
                                                    ≥▼
                                            is in list? thing
                                if
                         then
                                                              get litem *
                                                       list
                                                              get values
                                        set n to
                                                      0
                                then
                                        o add items to list list
                                                                   get values
                                 else
                                                           item
                                                                   get item *
                         get n 🔻
               result
```

To calculate the number of non-zero duplicates in a list, we build up a temporary list of values we have seen already as we traverse the input list, adding 1 to a return count \mathbf{n} each time we get a hit.

Paint_buttons

```
o to paint_buttons button_table
                                     invalid_ranges
    set global invalid_range_cells v to C call collect_invalid_cells v
                                                       invalid_ranges
                                                                         get global invalid_ranges •
                                                          range_cells |
                                                                        get global range_cells •
     for each cell_button in list ( get global cell_buttons v
          o initialize local cell to
                                                           get cell button •
                                                                               index ()
           initialize local button to
                                      select list item list
                                                           get cell_button
                                                                                index [
               set Button. BackgroundColor •
                                of component
                                                  get button *
                                                         is in list? thing get cell
                                                                                       list get global invalid_range_cells
```

This procedure first builds a global list invalid_range_cells using procedure <u>collect_invalid_cells</u> based on the global lists <u>invalid_ranges</u> and <u>range_cells</u>. It then examines each **cell_button** in the global list <u>cell_buttons</u>, breaking out its **cell id** and its **button** component and setting the button's BackgroundColor to red if the cell id is in the global <u>invalid_range_cells</u>, or white otherwise. (This has the desirable side effect of clearing the yellow highlight of a cell button that was just changed.)

Global invalid ranges

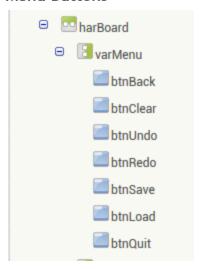
Global invalid_range_cells

Collect_invalid_cells

```
to collect_invalid_cells invalid_ranges
                                             range_cells
         initialize local invalid_cells to 🕻 🧿 create empty list
result
                   initialize local (range) to
                     initialize local cell to
                    for each (range_cell) in list ( get range_cells •
                         set range v to select list item list get range_cell v
                                          select list item list get range_cell •
                                                                                   index ( 2
                         if
                                               is in list? thing 📜 get range 🔻
                                                                               list 🔰 get [invalid_ranges 🔻
                                     and 🔻
                                                      is in list? thing 🌹 get cell 🔻
                                                                                    list get invalid cells
                                add items to list list get invalid_cells •
                                                                                item 🕻 get cell 🔻
               result
                        get [invalid_cells *
```

Invalid cells are the cells whose cell ids are in invalid ranges. We traverse the given list of range_cell pairs, extracting the range and contained cell id. If the range is not already in the list of invalid ranges and if the cell is in that range, we add it to the returned list.

Menu Buttons



When btnBack Clicked

```
when btnBack .Click
do call resume
```

<u>Resume</u>

```
to resume

do set varMenu . Visible to false 
set harDigits . Visible to false
```

The Back button just calls the <u>resume</u> procedure, which hides the Menu and Digits Arrangements.

When btnClear Clicked

```
when btnClear · .Click

do call handle_digit · n (0)

call resume ·
```

The Clear button hides the Menu and acts as if a 0 digit had been entered for a cell value, using procedure handle_digit.

When btnUndo Clicked

```
when btnUndo .Click
do
     🔳 if
                  length of list list 🖟 get global undos 🔻
     then
                                                          get global redos
            call push ▼ item
                                get global game *
                                                    to
                             Enabled ▼
               btnRedo ▼
                                               true
            set (global game 🔻 to (
                                    call pop r from
                                                       get global undos
                resume *
            call display validated game -
```

The Undo button is part of a pair of buttons that let you undo moves (btnUndo), and redo undone moves (btnRedo). They work off the game table, which reflects the current state of the game through the current values of all cells, and a pair of stacks,

- Global undos
- Global redos

During game play, at each move a copy of the <u>game</u> board is pushed onto stack <u>undoes</u>. When a move is <u>undo</u>ne, the game board is <u>push</u>ed onto the <u>redos</u> stack, then the prior value of the game board is popped off the undos stack. Changing the board with an undo operation requires complete revalidation of the board using procedure display validated game.

```
to pop
            from
         initialize local (top) to
                                    create empty list
result
                                    length of list list get from
               initialize local 1 to
              do
                    🔳 if
                                get 🔽
                          set top to copy list list
                                                          select list item list
                    then
                                                                              get from •
                                                                     index
                          remove list item list
                                                get from *
                                       index
                                                get 🔽
                        get (top *
               result
```

Popping from a list is done off the high index end.

Display_validated_game

```
to display_validated_game

do call display_game  Game  get global game 

call validate_all_sums 

call paint_buttons 

button_table  get global button_table 

invalid_ranges  get global invalid_ranges 

get global invalid_ranges
```

We first call <u>display game</u> to show the <u>game</u> board.

We then call validate_all_sums to check all ranges and fill global <u>invalid_ranges</u>. We then call <u>paint_buttons</u> to highlight bad cells.

Validate all sums

```
to validate all sums
                                   🕻 🧿 create empty list
do
     set global invalid ranges ▼ to
     for each range sum in list
                                 get global range sums
          initialize local range to select list item list get range_sum
     do
                                                   index [
              🧿 if
                          call range is valid ▼ r
                                                    get range
              then
                     call remove *
                                       get global invalid ranges
                                 list
                              value
                                       get range *
                        add items to list list 📗
              else
                                               get global invalid ranges *
                                        item 🕽 get range
```

This procedure does brute force validation on every **range_sum** pair in global <u>range_sums</u>. We call <u>range_is_valid_against the **range** in each **range_sum**.</u>

If the **range** is valid, we remove it from the global list <u>invalid_ranges</u>, otherwise we add it to the list <u>invalid_ranges</u>.

Gallery link

ai2.appinventor.mit.edu/?galleryId=5770654380064768

Other projects

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