Algorithm 1 Widget location array creation

Set a value for the threshold th[1]
Load the screenshot and all the widgets
Initialize as zeros an array of size $(N_{widgets}, 5)$ where to save the detection[2]
for Each widget **do**
  Calculate the Correlation Coefficients ($ccoeff$) for the widget on the screenshot
  if The widget is Datasets[3] **then**
    $th \leftarrow 0.7$
  else
    The threshold stays the same
  end if
  Find how many coefficients are above th
  if There is at least one coefficient greater than th **then**
    Order the coefficients so that we can save them in descendent order
    Initialize a location vector for each axis to save where the current widget is already been found
    for Each coefficient greater than th **do**
      if The location of the found widget is far from previous ones **then**
        Save the widget's presence (dimension 1), location (dimension 2), output shape (dimensions 3 and 4) and the found $ccoeff$ (dimension 5) in the previously zero-initialized array
        Append the found widget axis locations to the axis location vectors
      end if
    end for
  end if
end for
$j \leftarrow 0$
Calculate $N_{detected\ widgets}$
while $j < N_{detected\ widgets}$ **do**
  Find locations of the detected widgets
  if Some widgets overlap **then**
    Find which one of the overlapping widgets has the highest $ccoeff$
    Delete the match for the widgets with lower $ccoeff$'s
    Calculate $N_{detected\ widgets}$
  end if
  if The widget deleted wasn't the j-th **then**
    $j \leftarrow j + 1$
  end if
end while

[1]The standard chosen value for the threshold is 0.78 as it seems to be the perfect balance between finding most widgets and not finding too many.

[2]The first dimension of this array is useful since there may be repetitions of the same widgets in the same screenshot. If the spot for the widget is already taken the algorithm will look for a previous one and save in number form how many cells before the actual detected widget the information is stored.

[3]This widget has a lower resulting Correlation Coefficient, so the threshold needs to be lower compared to the other widgets.

Algorithm 2 Get widgets from image

Import the array created by the "Widget location array creation" algorithm
Find the widgets that have a non-zero element in the first dimension of the array
Initialize an empty list
for Each non-zero element **do**
  Append the name of the identified widget to the list, keeping track whether a widget cell is used as a place holder for later ones (dimension 1) or not
end for

Algorithm 3 Get widget size from image

Load the screenshot as a grey-scale image
Apply Gaussian blur to the screenshot
Identify circles in the screenshot through Hough Circle Transform and extract their *radii*
$widget\ size \leftarrow round(median(radii * 2) + 1)$

Algorithm 4 Find circle intersection

Initialize the values for the center, the radius of the circle
Create a binary image with the same size of the screenshot that contains a white circle of the given center and radius
Perform pixel-wise *and* between the circle image and the thresholded screenshot image
for Every found cluster of points **do**
  Find which pixels are white and apply the mean to the cluster to find the position of their center
end for
Given the points found at the previous steps calculate their angle relative to the center of the circle with the inverse tangent function

Algorithm 5 Get widget pairs from image

Import the array created by the "Widget location array creation" algorithm
Get the the detected widget size from the "Get widget size from image" algorithm
Find which widgets are detected in the screenshot through the widgets array
Initialize with zeros the link array
Load the screenshot
Binarize the screenshot through thresholding in order to highlight the links between widgets
Identify connected components in the obtained binary image and label them with integers
for Each identified widget **do**
  Find which labels are in the near vicinity of the identified widgets
  Create two arrays to keep track of which labels are identified to the right of the widgets and which to the left[4]
end for
for Each identified widget **do**
  Extract the connected components labels that are identified as incoming relative to the widget of interest
  for Each of the found components **do**
    Check if they are also identified as outgoing relative to the other widgets
    if There is exactly one outgoing link **then**
      Assert which widget has the outgoing link
      Save in the link array that there is a link between the two widgets
    else if There is more than one outgoing link **then**
      Create a binary image containing just the link of interest
      Use the "Find Circle Intersection" algorithm to find the link starting points
      for Each found starting point **do**
        $previous\ direction \leftarrow 180$
        $center \leftarrow (x, y)$ of the starting point
        while True **do**
          Use the "Find circle intersection" algorithm to find the new center and the new direction
          if $abs(abs(previous\ direction - new\ direction) - 180) < 20$ **then**
            $center = (x_{center} + 1,\ y_{center})$
          else
            $center \leftarrow new\ center$
            $previous\ direction \leftarrow new\ direction$
          end if
          if The center of the circle is very close to another widget **then**
            Assert which widget has the outgoing link
            Save in the link array that there is a link between the two widgets
            break
          end if
        end while
      end for
    end if
  end for
end for