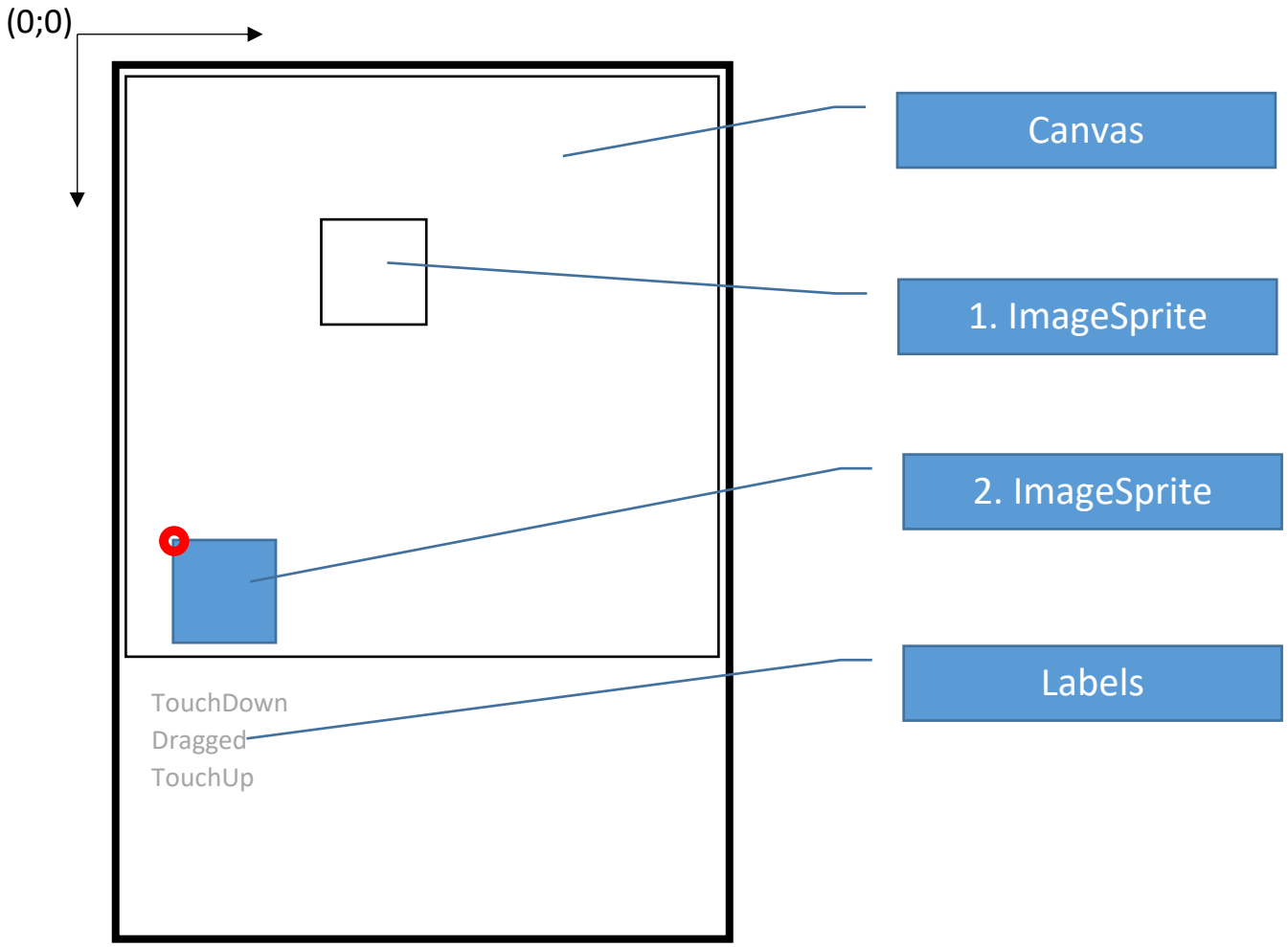
Managing a puzzle element

Level: Beginner

Time: 2x45 Mins

Step 1: Visual design



Picture1: Screen design

Step 2: Creating the Screen (for testing)

Building up the Game Area:

-Screen: Screen1, Center, Center, Icon: isr.PNG, Title: Build the DSL network

-Canvas: mainCanvas, 300px height, Fill parent… wide

-1. ImageSprite: place1, 45px *45px, X:90, Y:60, picture: place1.jpg

-2. ImageSprite: picture1, 40px*40px, X:5, Y:240, picture: icon1.png

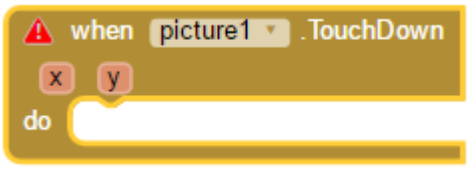
Creating the Information bar:
-Label: touchDown, Text: „”
-Label: dragged, Text: „”
-Label: touchUp, Text: „”
Explain it: What does the (x;y) coordination pair means in case of a planar shape (element)?

Step 3: Declaring the necessary global variables

Here, we have to focus on what we can accept as “the correct place” for the puzzle. This is going to be named as (endX;endY) variables. Furthermore, if an element is placed into a wrong place, it has to “move back” into the starting position which is going to be named as (beginX;beginY). These variables are going to be used in several other procedures, therefore they must be declared as global variables.

Picture2: Global variables

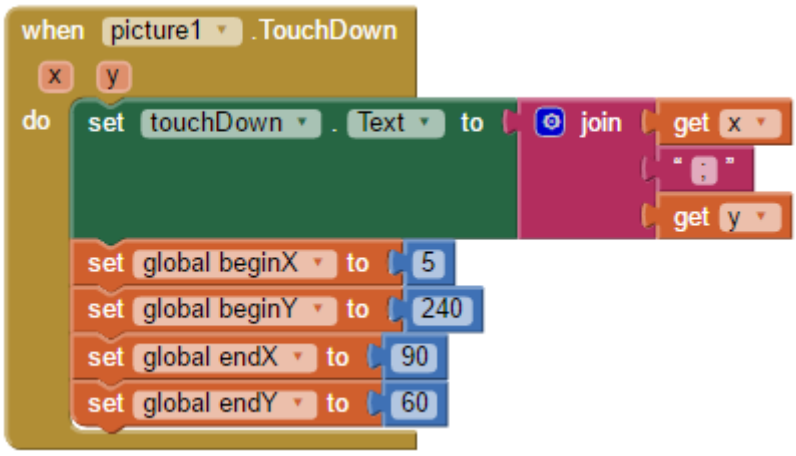
Find out: Declaring variables in this way is called camelCase. What does it mean exactly?



Step 4: Programming the "touchDown" event

The screen coordinate, where it was grabbed, is loaded into TouchDown label. These two data(x;y) are automatically there for us when the TouchDown event is called. (This information is not important for us, but we can check the screen resolution with the help of them). After this, we set the values of the global variables to the values of picture1 (beginX=5, beginY=240, endX=90, endY=60).

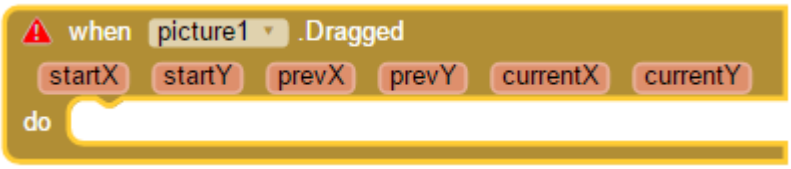
Explain it: The currently (grabbing picture1) set values are the same as the coordinates X and Y of picture1 in the Designer. Why is it so?



Picture3: The TouchDown event

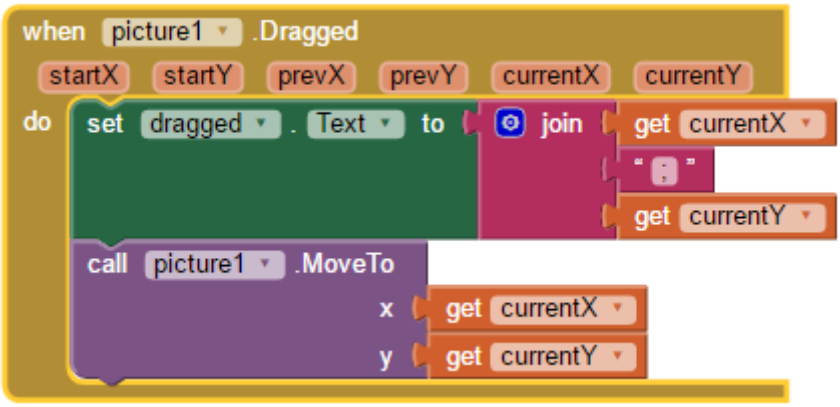
Test it: Check what coordinates are being displayed by clicking on the element. Try to find a coordinate that is almost outside of the Screen.

Explain it: Why doesn't we get a coordinate pair when we click on the Screen instead of the Puzzle element?



Step 5: Programming the "Drag" event of the element

That coordinates are loaded into dragged label where we are currently dragging the puzzle. These coordinates are automatically there when we call the event: Dragged (currentX;currentY). These coordinates are only helping our orientation. To move the element, we must call the MoveTo method. The element is going to move to the place which is represented by the (x;y) coordinates of this method. According to this, simply x=get(currentX) and y=get(currentY) parameters are needed to set.

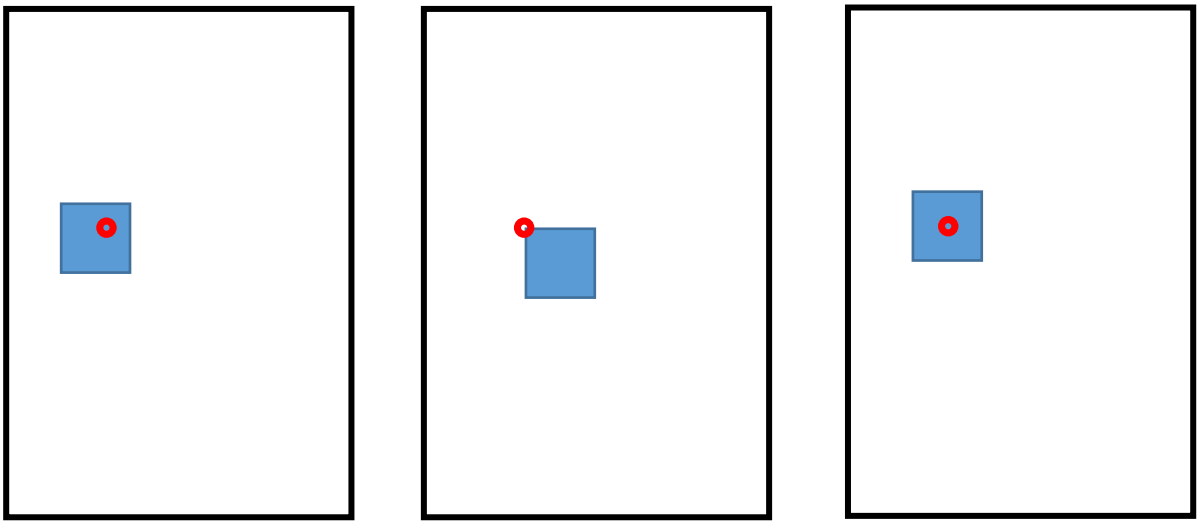


Picture4: Dragged event v1

Test it: Check what coordinates you get when you click on the element and drag it. Watch how the element jumps under your finger when you start moving it, by doing this you can't really see where you drag it.

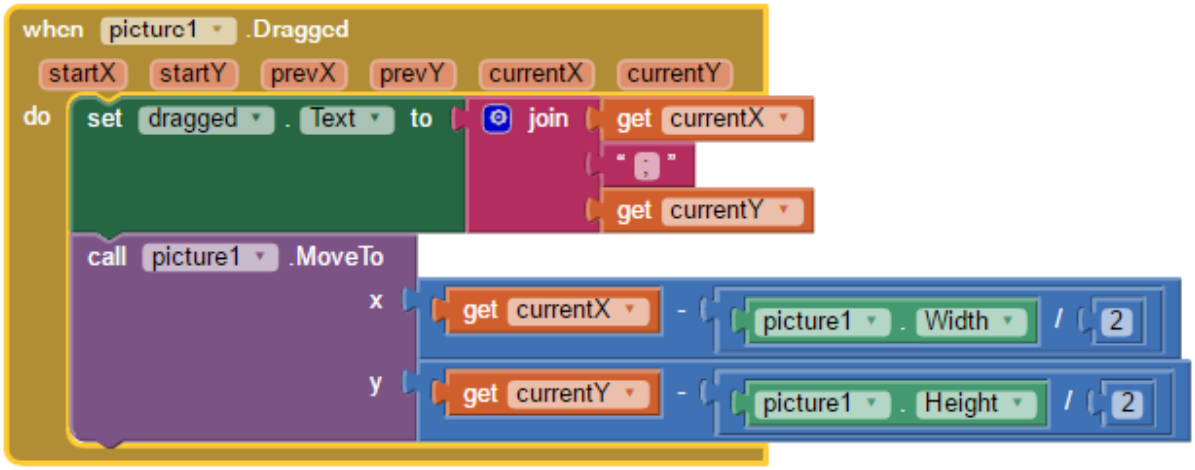
Think about the problem: What causes it?

The red circle indicates the position of a click. Its coordinates are (currentX,currentY). If you start dragging it, the MoveTo event starts positioning the top-left corner (check Picture5), which means, you cannot really see where the dragging goes. That is why dragging must be positioned into the middle of the Puzzle element, so not putting the element of the top-left corner to the current coordinate (currentX;currentY), but shifting it with half size of the element to **top-left**. So, x=currentX-(picture1.Width/2) and y=currentY-(picture1.Height/2).



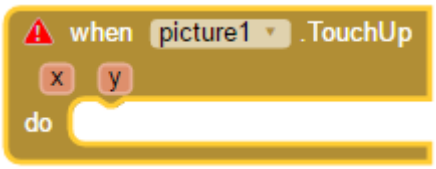
Picture5: Repositioning

Let's have a look at the code:



Picture6: Dragged event v2

Test it: Check if you click and drag the element you can clearly see it now.



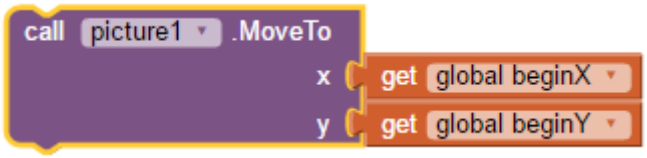
Step 6: Programming the "MoveTo" event

Think about it: What should we do in the "MoveTo" event?

The releasing coordinates are displayed to touchUp label.

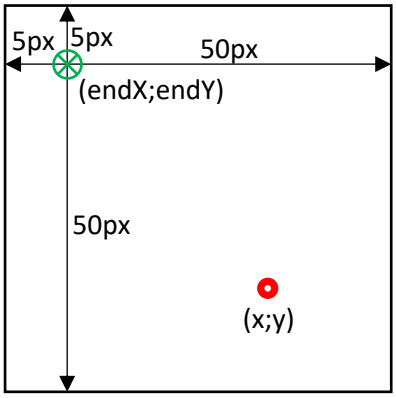
When the user releases the puzzle element at the correct coordinates (endX;endY) we accept that and in any other case we put the puzzle element into its starting position (beginX;beginY)

So to the false branch of the condition a MoveTo method comes with (beginX;beginY) parameters.



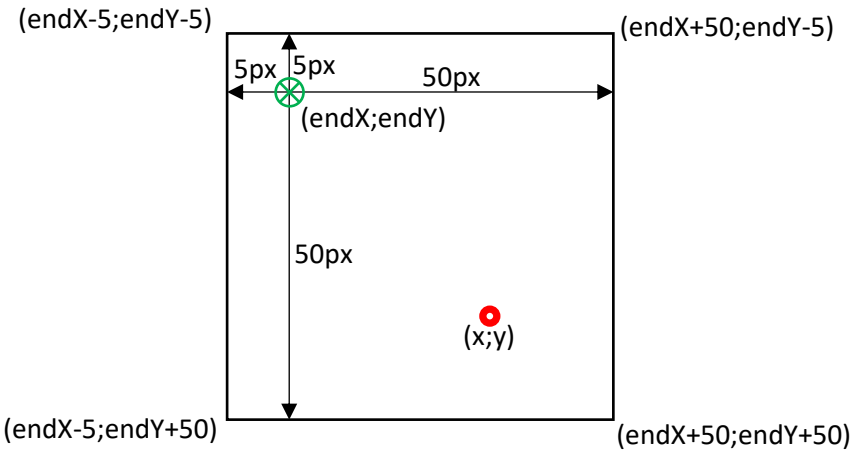
Picture7: Moving back

Since there is really a little chance that the user of the application will release the puzzle element at the exact X and Y coordinates, you have to make the correct position a little bit bigger. In other words, the destination coordinate should examine a given environment. It can be achieved that the size of the destination place must be increased. Based on our experiences the following square should be examined.



Picture8: Acceptable size of range

The green X coordinates are the ones that we accept. So, the releasing of the puzzle is at the correct place when you still have 5px to the left, 5px to the top, 50px to the right and 50px to the bottom.

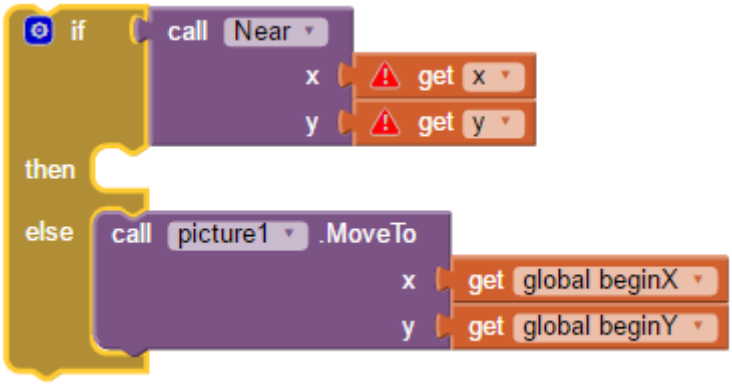


Picture9: The acceptable coordinates of the square

To finish the method, the only thing to do is to get to know the coordinates of the red point (release). Luckily, this data is stored in the TouchUp's event (x and y). To see clearly this code, another event is made, which is called by the releasing coordinates from TouchUp event management. This is how the algorithm is going to work:

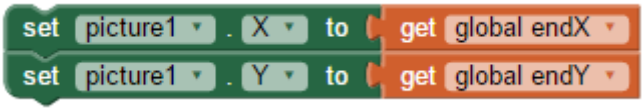
((releasing X coordinate smaller, than position X + 50px **and**
releasing X coordinate bigger, than position X - 5px) **and**
(releasing Y coordinate smaller, than position Y + 50px **and**
releasing Y coordinate bigger, than position Y - 5px))

Picture 10: Checking the acceptable field with the event



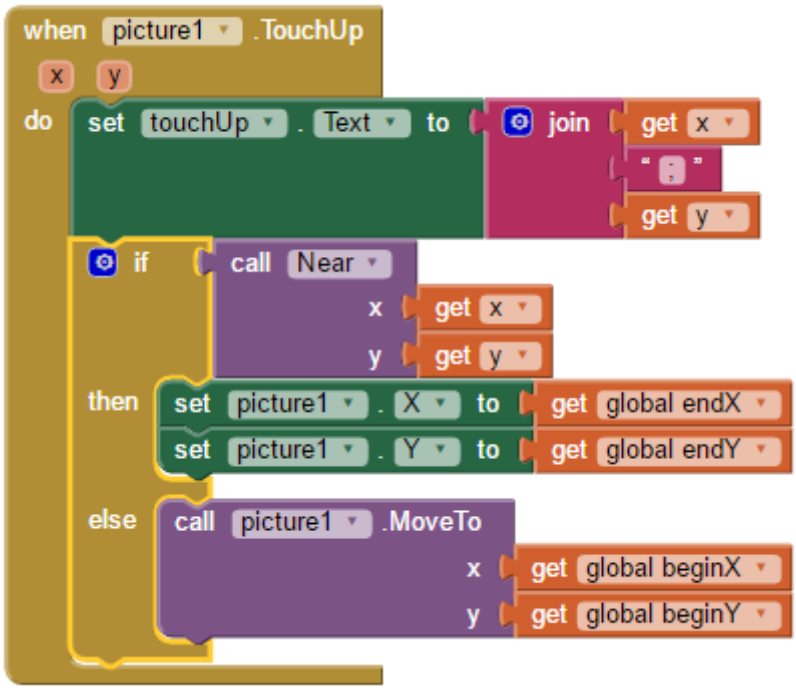
Picture11: Calling the Near event

If this criterion is met, then we simply need to put the element into its destination which is at the (endX;endY) coordinates:



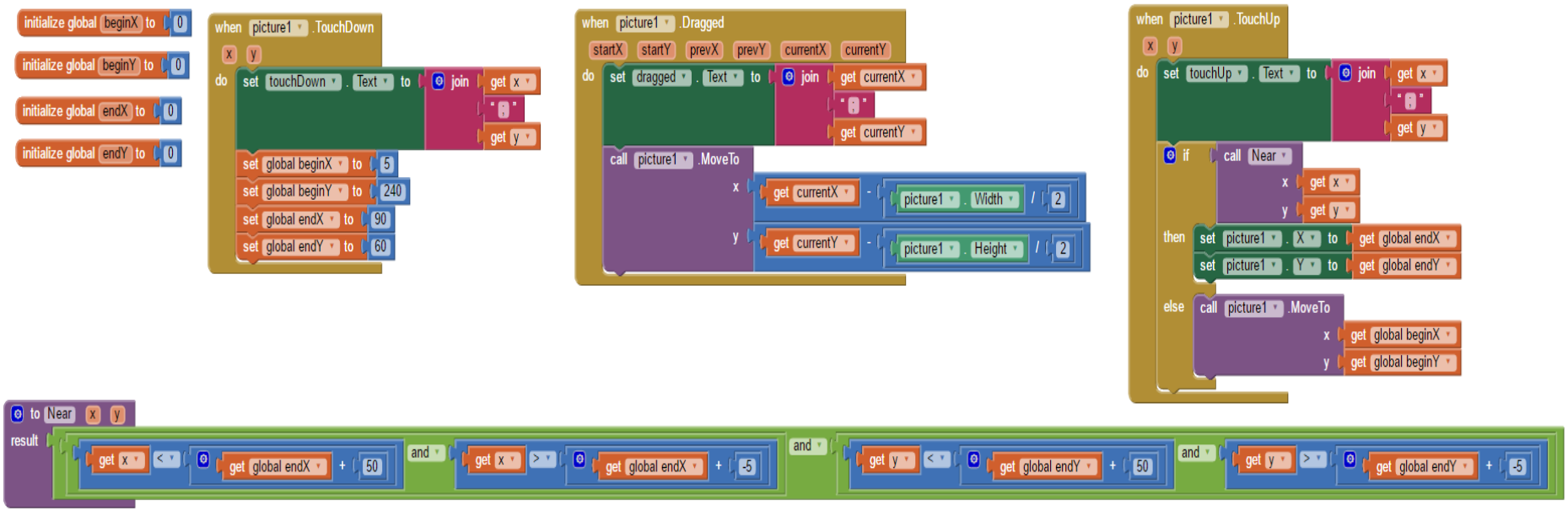
Picture12: Positioning the element into its place

At last let's see the event management



Picture13: TouchUp event

Full code:



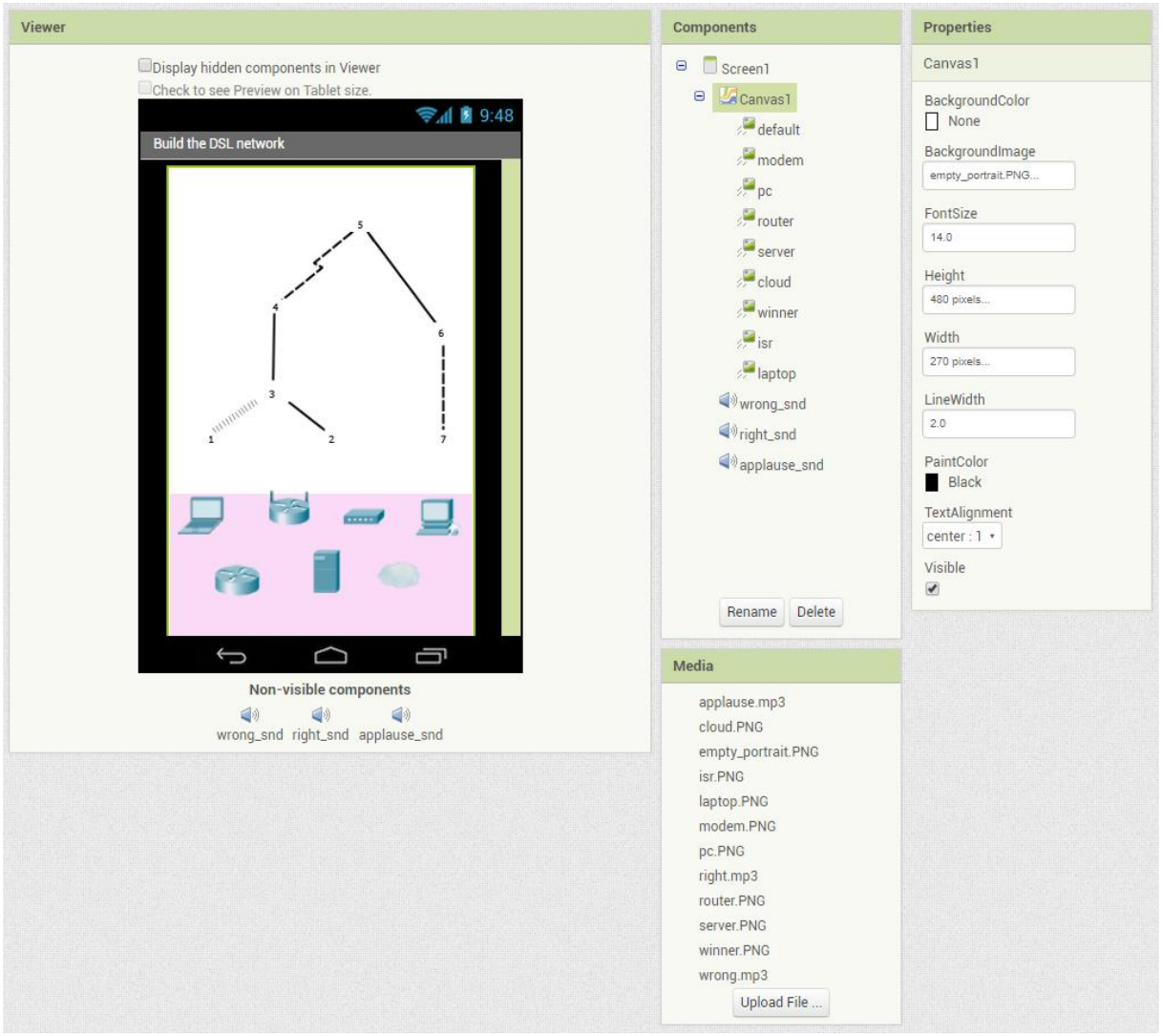
Picture14: The whole block of the app

The networkPuzzle app

Level: Intermediate

Time: 4x45 Mins

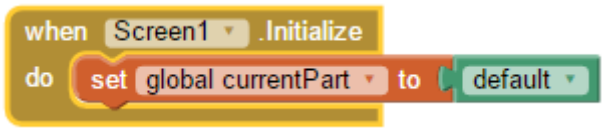
Designer:



Picture1: The app in designer view

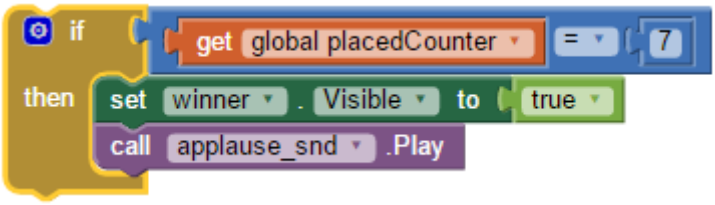
Think about it: How our tasks change in case of more elements?

1. It is always has to be known what elements are grabbed/moved/released -> a variable, named currentPart, is necessary, which is set to a default value in the beginning of coding.



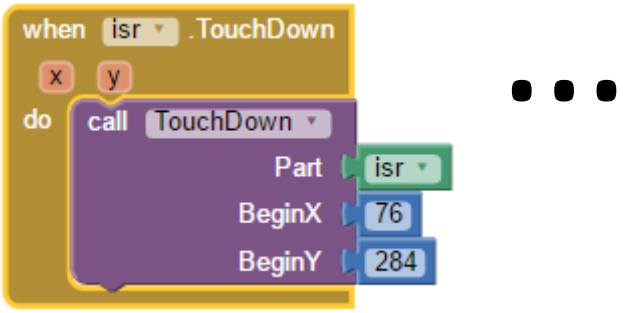
Picture2: Setting currentPart variable in the beginning

2. It is also has to be known, which element was put to its place successfully. -> a variable, named placedCounter, is necessary, which is examined after an element is dragged to its place. If the value of placedCounter is the same as the number of the elements, then we are ready.



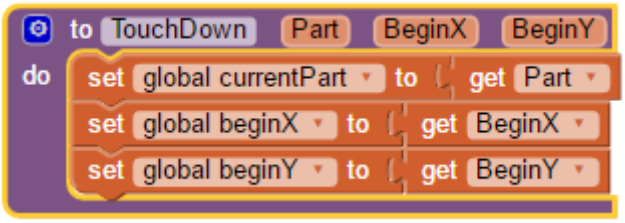
Picture3: Checking the status of the puzzle

3. Because more than one element is required to perform the same activities, so because of the principle of decision postponement, the event managements are needed to be coded as simple as they are possible. And the common parts should be parametrized in different methods with variables.



Picture4: TouchDown event manager for each element

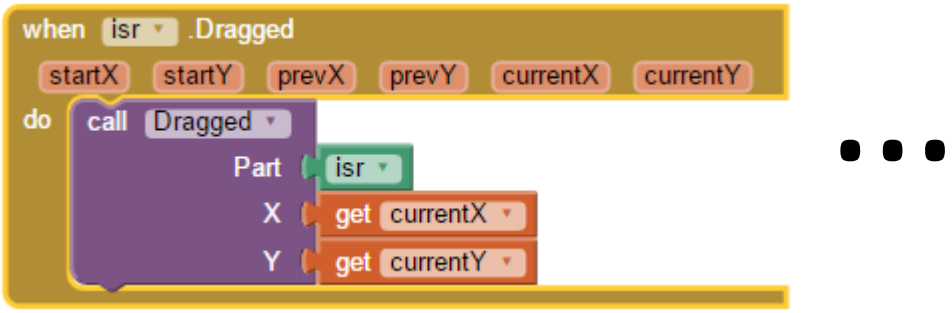
(It is recommended to decide about the coordinates with the help of the Auxiliary table at the end of this document)



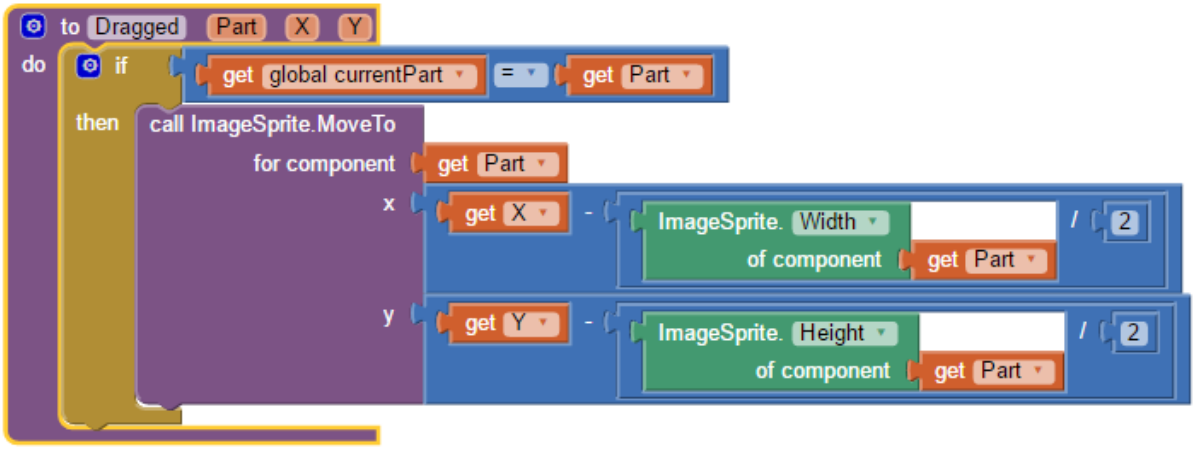
Picture5: The same TouchDown method for each element (needed to be coded only once)

Here the TouchDown event management sets the (beginX;beginY) coordinates and element name. The (endX;endY) are not here because they are going to be set only in TouchUp event management.

For making difference between global variables and variables as given (and received) parameters, the last ones start with capital letters.



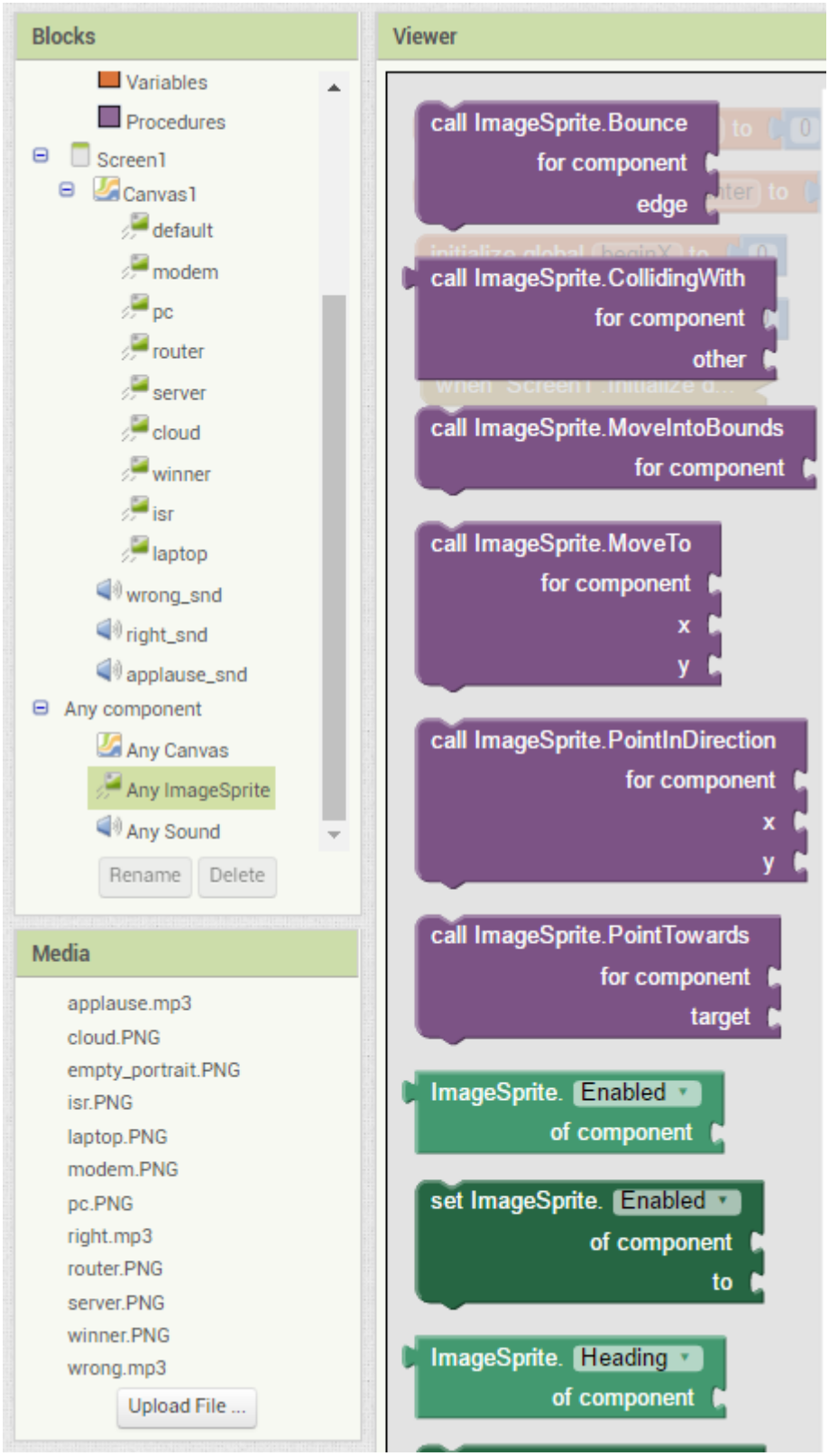
Picture6: Dragged event management has to be coded for each element



Picture7: The same Dragged method for each element (needed to be coded only once)

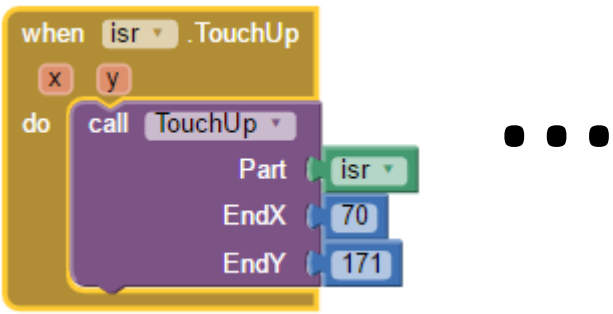
The interesting part of Dragged method is, that it knows by item ID (Part) which item it must deal with. MoveTo method “component” parameter

serves for this. Methods like this (can be parameterized with component) are available from Any component/Any ImageSprite menu.

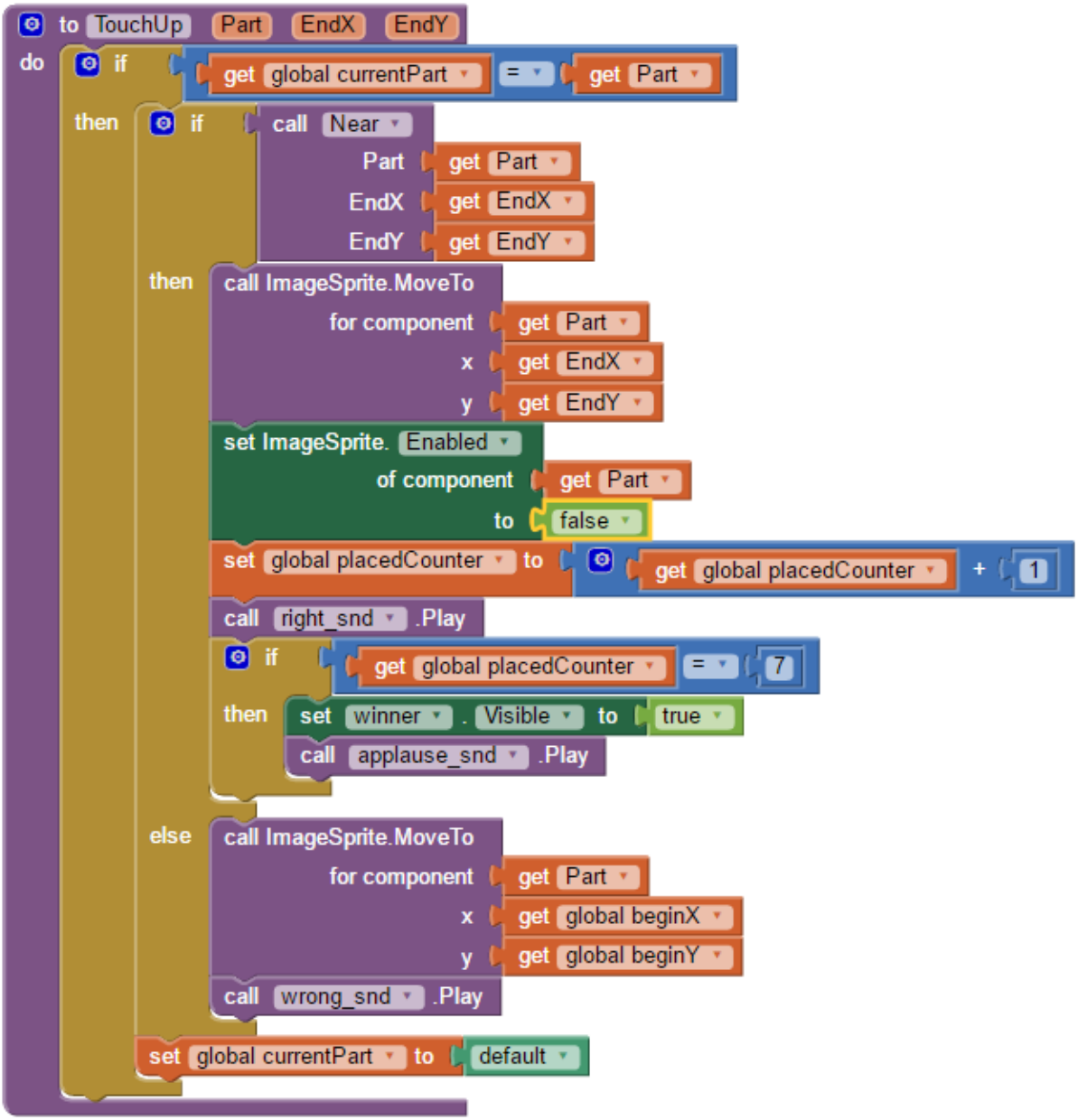


Picture8: Reaching ImageSprite based on component

Finally, “release” event management:

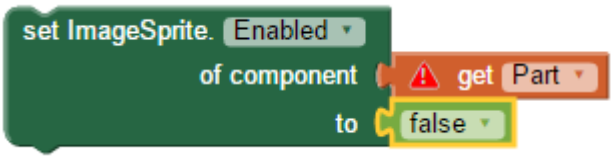


Picture9: TouchUp event management has to be coded for each element



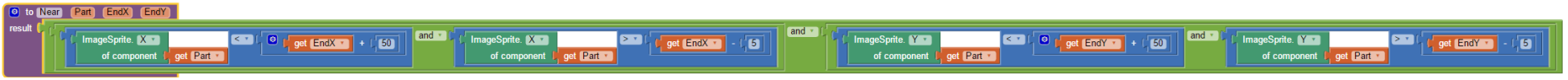
Picture10: The same TouchUp method for each element (needed to be coded only once)

If releasing happens at the right place, then after dragging the element to its destination place, enabling further moving must be set. See the code below:

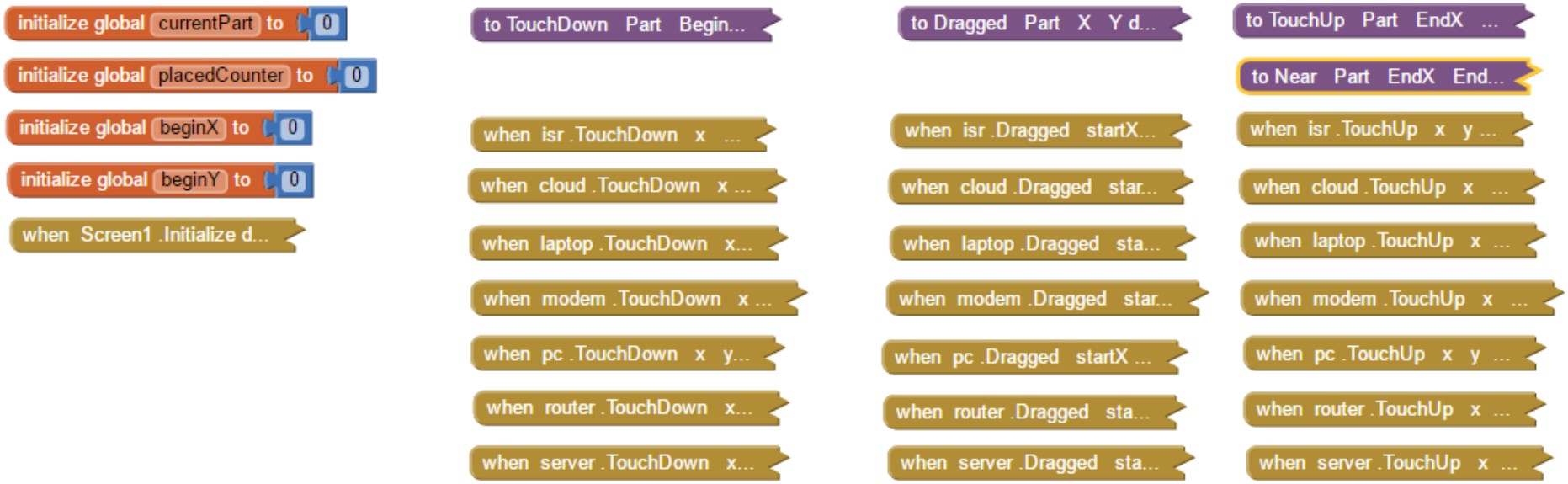


Picture11: Enabling moving an element from its place

If there is a release, then at the end of the event management the value of currentPart must be set to default.



Picture12: Function Near



Picture13: The prototype list of the app

Auxiliary table:

No.	Item name	Pic	Size (px)	beginX	beginY	endX	endY
1	isr	isr.png	40x40	76	284	70	171
2							
3							
4							
5							
6							
7							