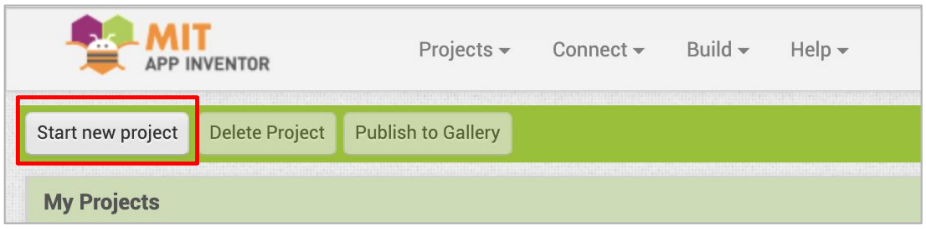
Personal Image Classifier: Part 3

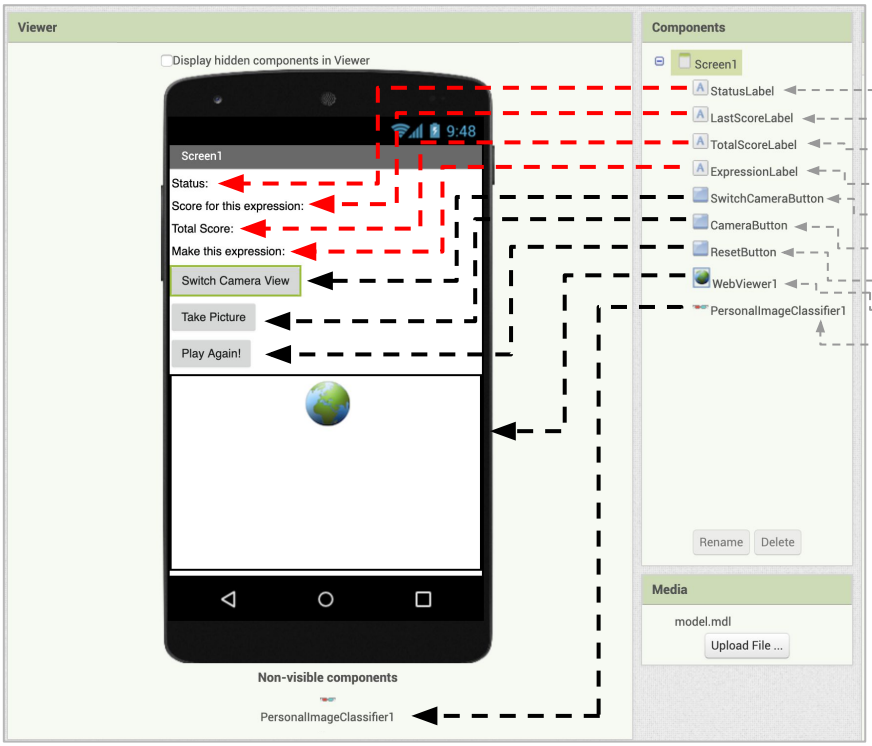
In this lesson, you will make an expression match game using MIT App Inventor

1. Go to **http://ai2.appinventor.mit.edu** and login. You will need to make an account if you do not have one.

2. Once you are logged in, click “Start new project” in the top left corner. Name your project **ExpressionMatch** and click “OK”.



3. Below is what you will create in the Designer. Review what each component is used for. You can try to create this user interface yourself and then skip to step 17, or follow the instructions on the following pages.

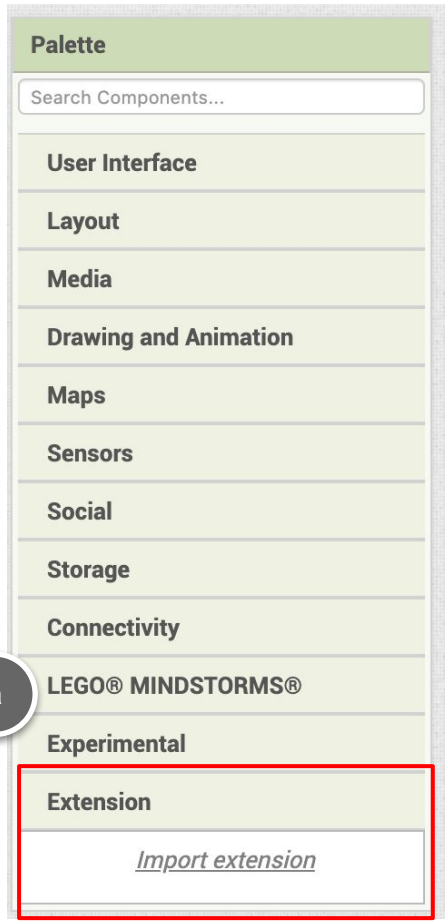


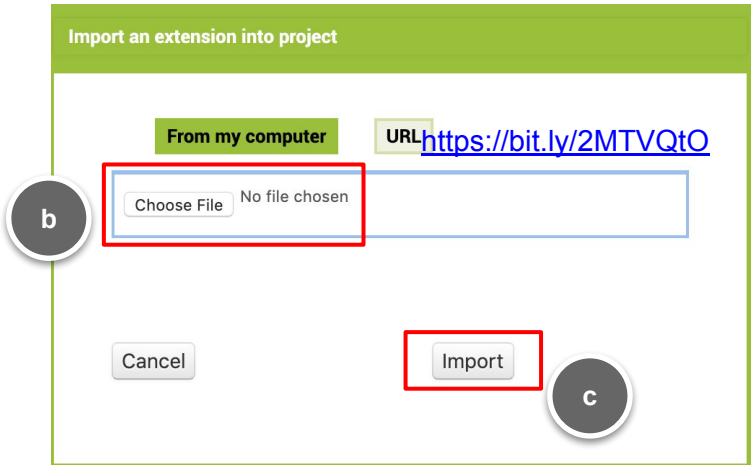
- Displays Classifier status
- Displays score for current expression
- Displays total score
- Displays expression user should make
- Allows user to toggle camera direction
- Takes picture and classifies image
- Resets game when all expressions matched
- Displays video for Classifier
- Classifies images based on trained model

NOTE: All the Buttons should be invisible to start, so uncheck the *Visible* property for those components.

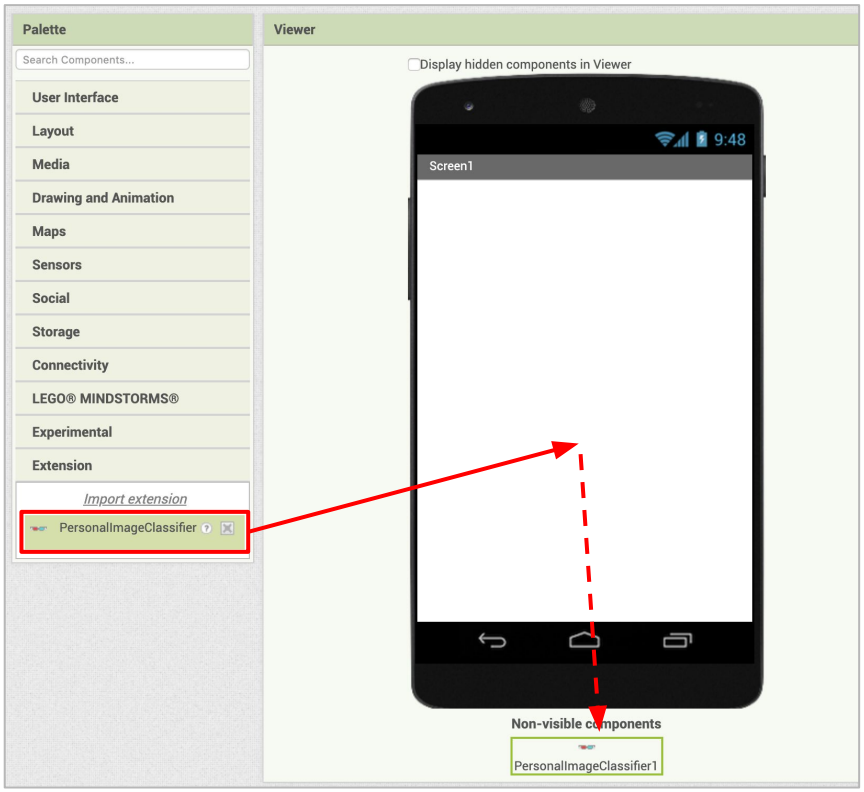
5 Download the PIC extension file from https://bit.ly/2MTVQtO) to your computer.

5 In the Designer, find “Extensions” in the Palette and click on “import extension”. Import the “PersonalImageClassifier” extension you just downloaded.



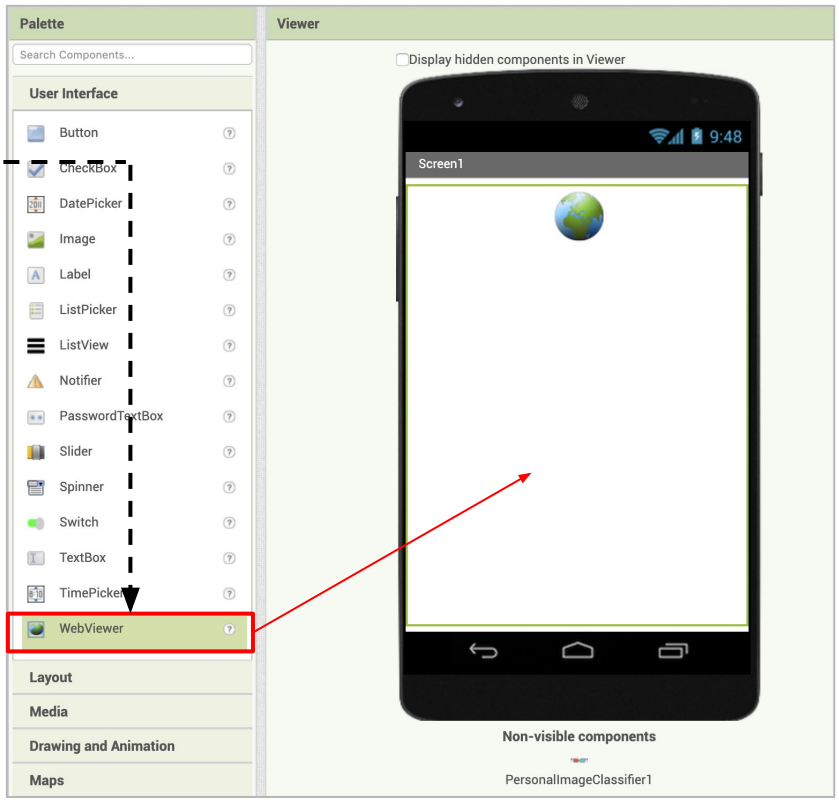


6 Then, drag the extension into the Viewer to add it to your app. It will appear at the bottom of the screen, - - - → because it is a non-visible component.

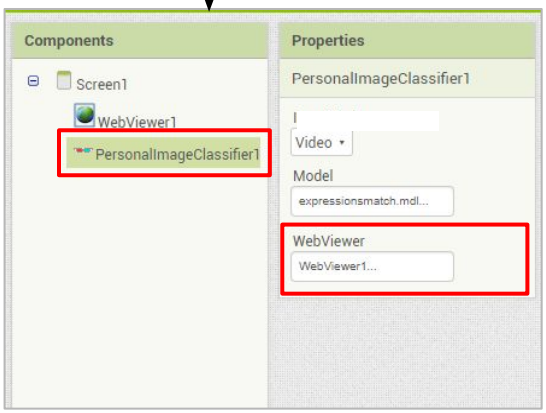


This extension requires a WebViewer component and a model file to work.

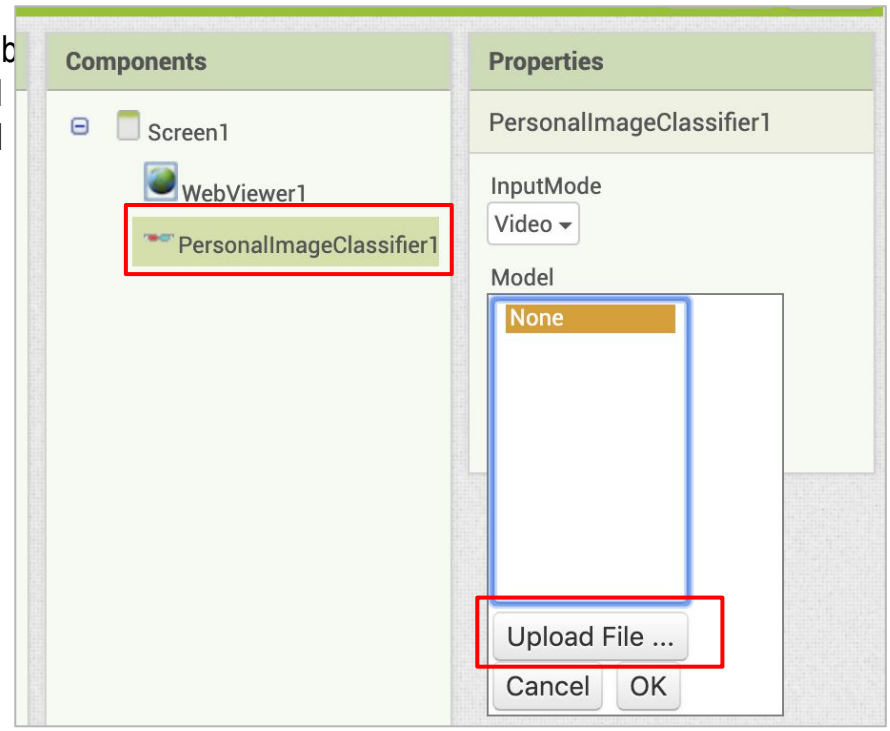
7 Drag in a WebViewer component from the User Interface drawer. Change its *Width* and *Height* properties to **"Fill Parent"**.



8 Click on **PersonalImageClassifier1** And select **WebViewer1** as the *WebViewer* property.



9 Upload the model you just trained b clicking on the *Model* property and selecting the model you made and downloaded.

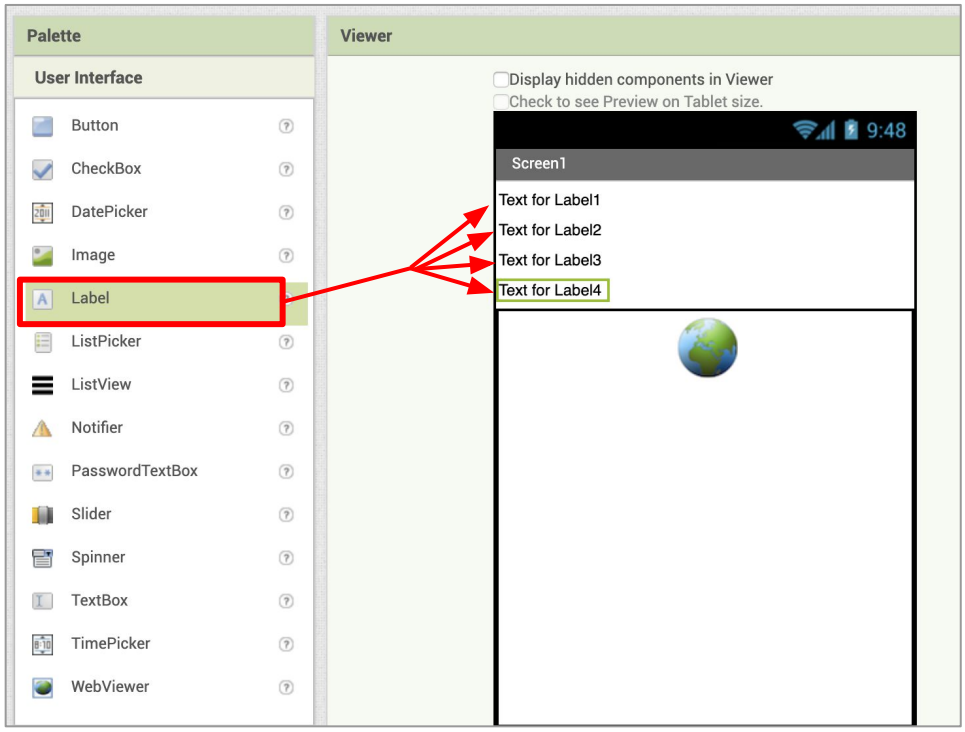


Now that you have set up the extension, you can start working on the app. Start by adding some labels.

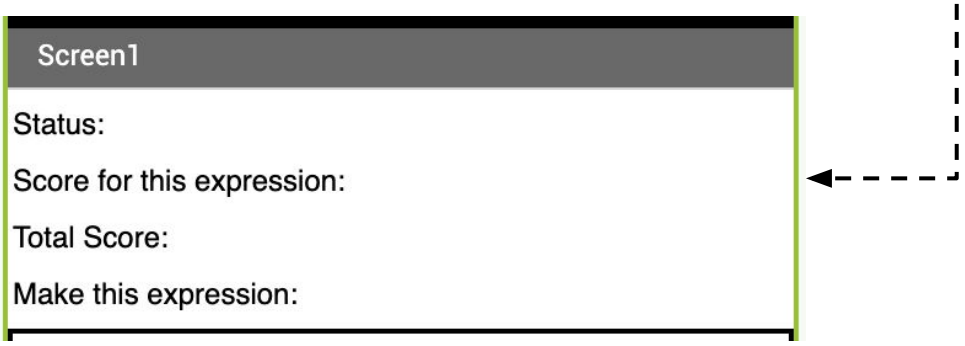
10 In the Designer, drag in 4 Labels.

11 Rename them:

- StatusLabel
- LastScoreLabel
- TotalScoreLabel
- ExpressionLabel



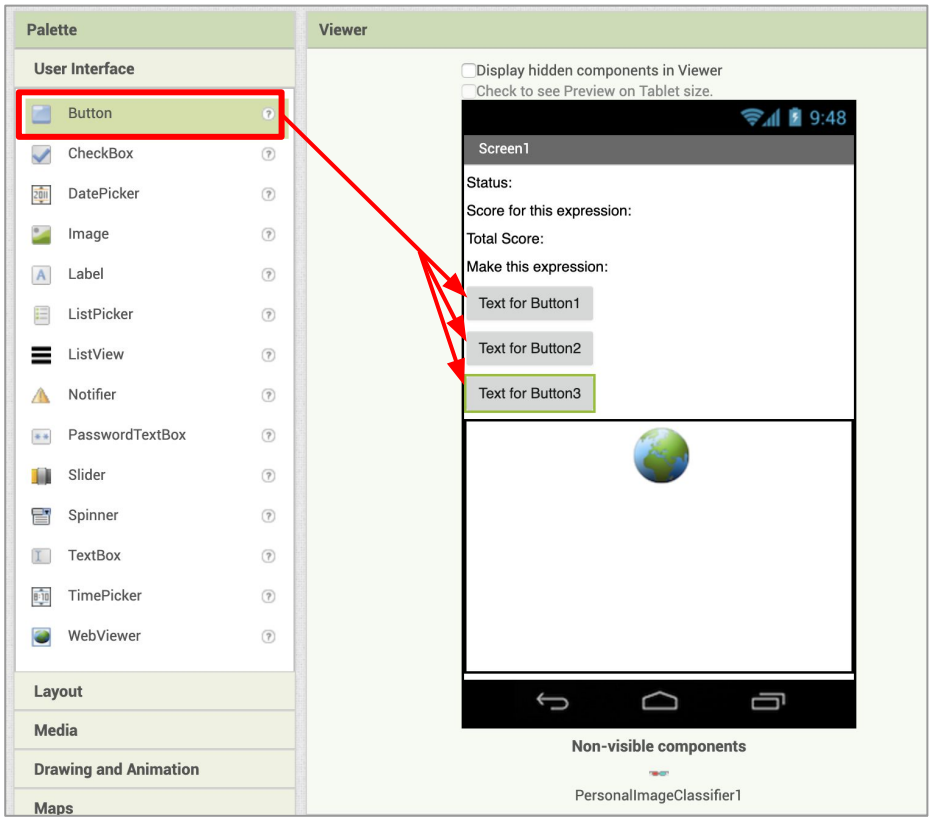
12 Change the *Text* property for each Label so your interface looks like this.



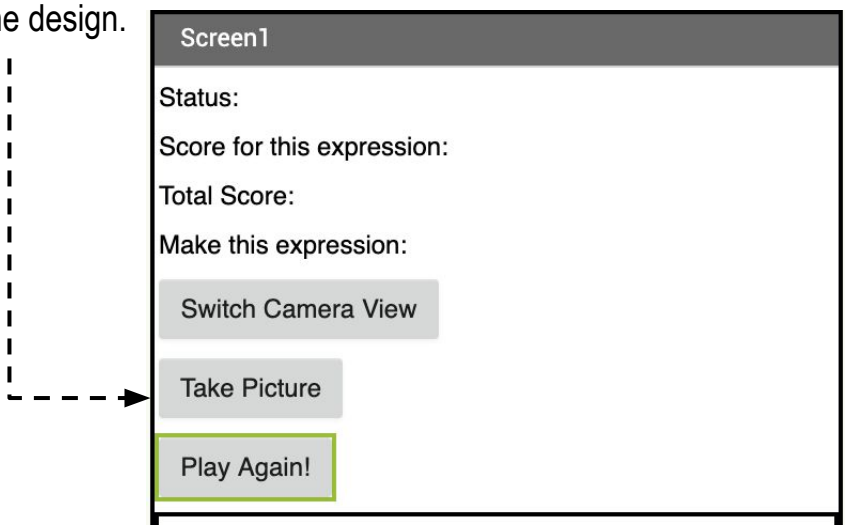
13 Drag in 3 Buttons.

14 Rename them:

- SwitchCameraButton
- CameraButton
- Resetbutton



15 Change the *Text* property for each Button so your interface looks like this. This is a basic layout. Feel free to move components, and change their color and size to put your own style into the design.

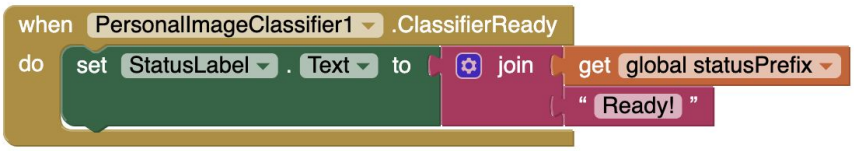


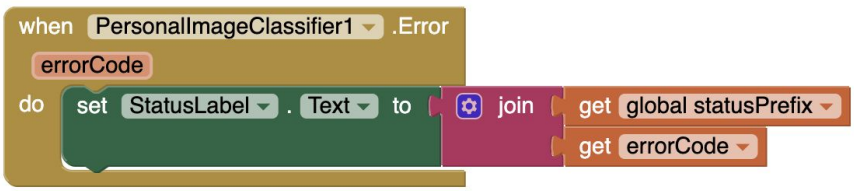
16 For the three Buttons, uncheck the *Visible* property so they are hidden when the app starts.

17 Now switch to the Blocks Editor to start coding the behavior for the app.

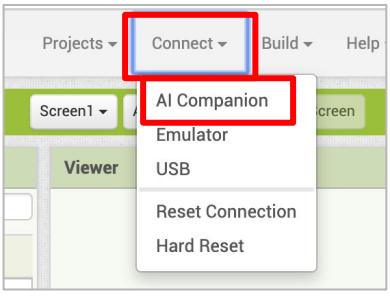
18 Add four **initialize global** variable blocks to set up the prefixes for each of the Labels. You will use these text prefixes and join them with the information to be displayed in the appropriate Label. Remember to add a space at the end of each text block.

19 Add the **PersonalImageClassifier1.ClassifierReady** event block to update the status label when the model is ready. Also add the **PersonalImageClassifier1.Error** event for when an an error is received, and update the status label with the error code received.





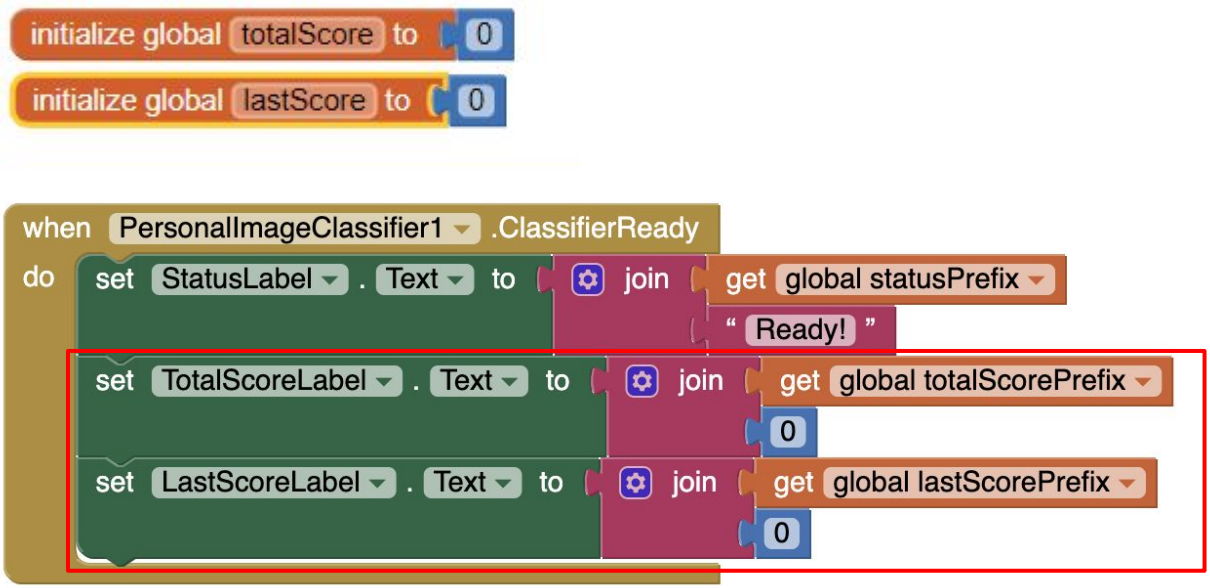
20 Connect your mobile device using the AI2 Companion now. You should see the status label change to **"Status: Ready!"** when the model is done loading.



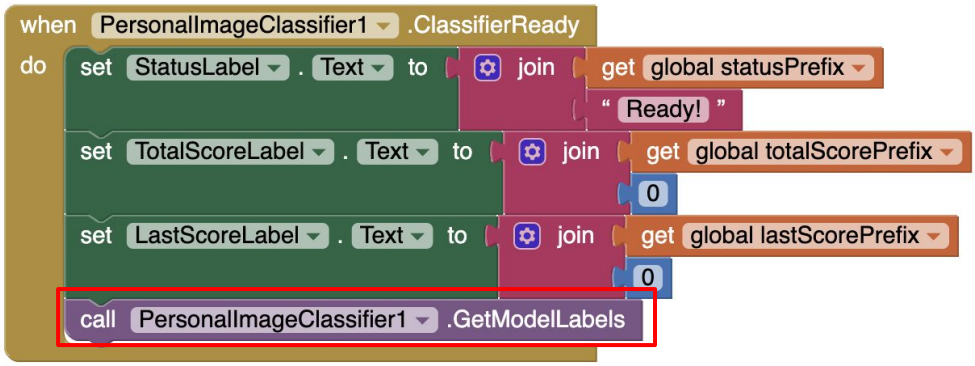
NOTE: If your status label does not change, your mobile device may not have the correct hardware to run the extension.

Next, you need to set up the scoring system for your game and get the labels (the expressions you entered) from your model so that you know what expressions to tell the player to make.

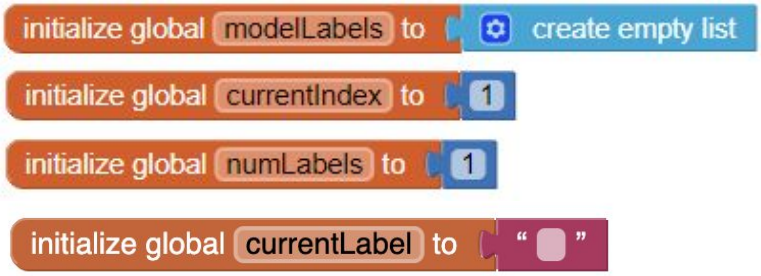
21 In the Blocks Editor, add two variables to keep track of the total and last scores. Also update the **ClassifierReady** event to initialize the Text for the corresponding Labels.



22 To get a list of the labels (expressions) from your model, you can use the **GetModelLabels** block. Add this call to the end of the **ClassifierReady** event block.

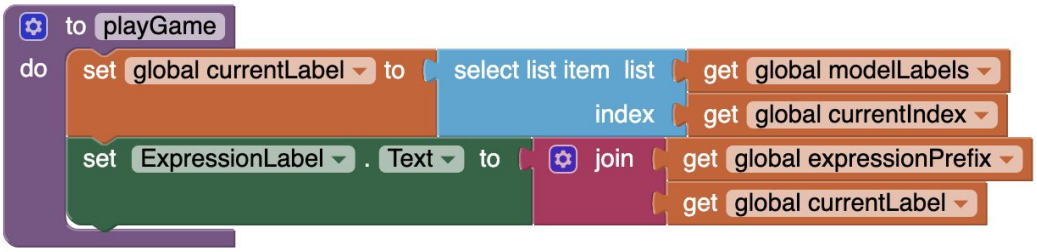


23 Set up 4 more variables: one to store the list of labels, one to store an index for this list, one to store the number of labels the model has, and one to store the current label (expression).



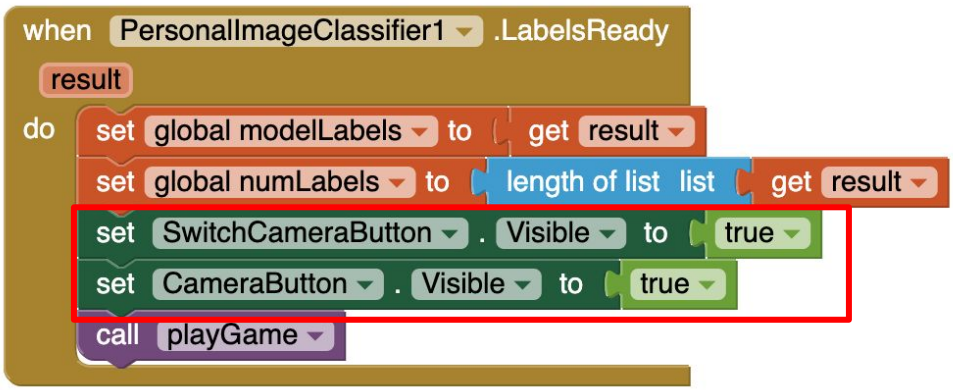
24 **GetModelLabels** triggers an event, **LabelsReady**. The result returned is the list of model labels, so set the **modelLabels** variable to the result. Set **numLabels** to be the length of that list.

25 You'll want to make this prompt multiple times, so put it inside a procedure called **playGame**. **playGame** will use the **currentIndex** variable to select an expression to prompt the user with, and will update the **ExpressionLabel** with the new expression to make. The first call to **playGame** will be at the end of the **LabelsReady** event.

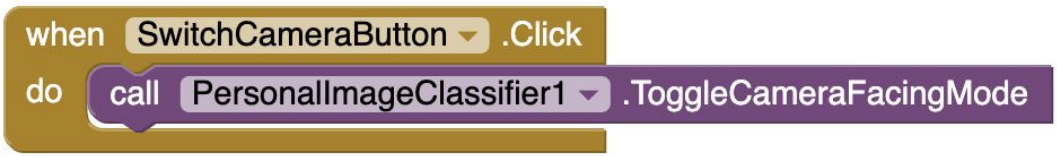


Once you receive the labels from the Classifier, you want the game buttons to appear.

26 Update the **LabelsReady** event to turn on the **SwitchCameraButton** and **CameraButton's** visibilities, so that the player can only take pictures when the game is ready to start.



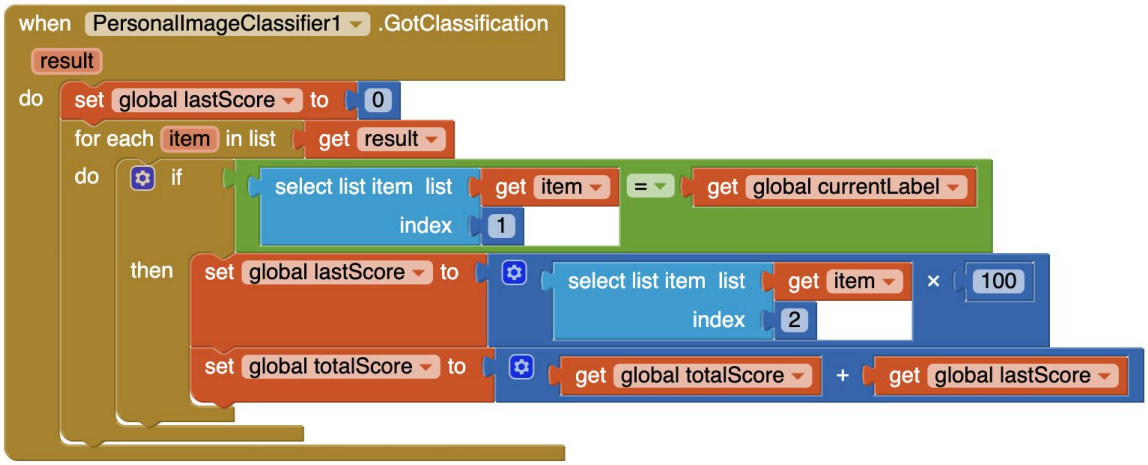
27 Add functionality to the Buttons. **SwitchCameraButton** should call **ToggleCameraFacingMode** to allow the user to switch camera views. **CameraButton** should call **ClassifyVideoData**, which asks the model to classify whatever is in the video feed from the **WebViewer**.



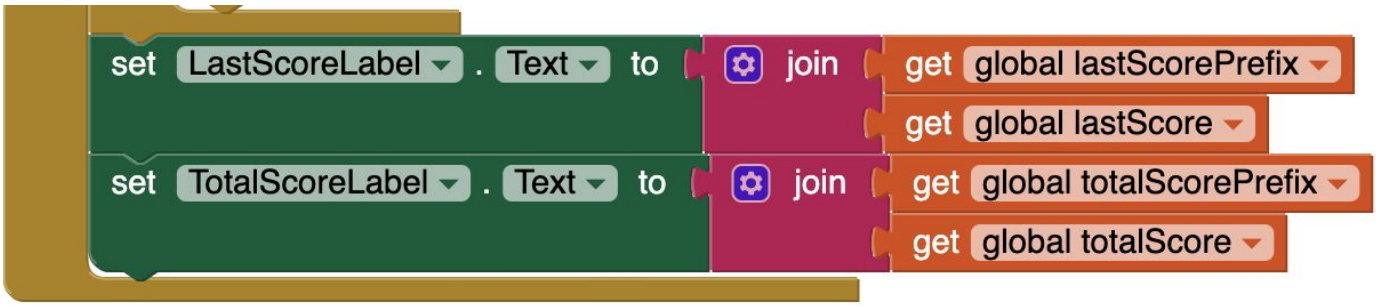
It's time to add the main logic of the game. This includes updating the score after taking pictures and determining when the game ends. You can do this in the **GotClassification** event, which is triggered after an image is classified (when **ClassifyVideoData** is called).

The classification results are a list of lists with what the Classifier thinks the image is. Each sublist contains two items - the guess, and the confidence level. (eg. [[Smile 0.8] [Surprised 0.3] [Sad 0.1]].

28 To score the player, first, reset the player's last expression score to 0. Then, use a **for each item in list** block to iterate through the classification results. If the expression is in the result list, set the **lastScore** to the confidence level x 100 (to make it larger than 1), and then add that score to the **totalScore**.

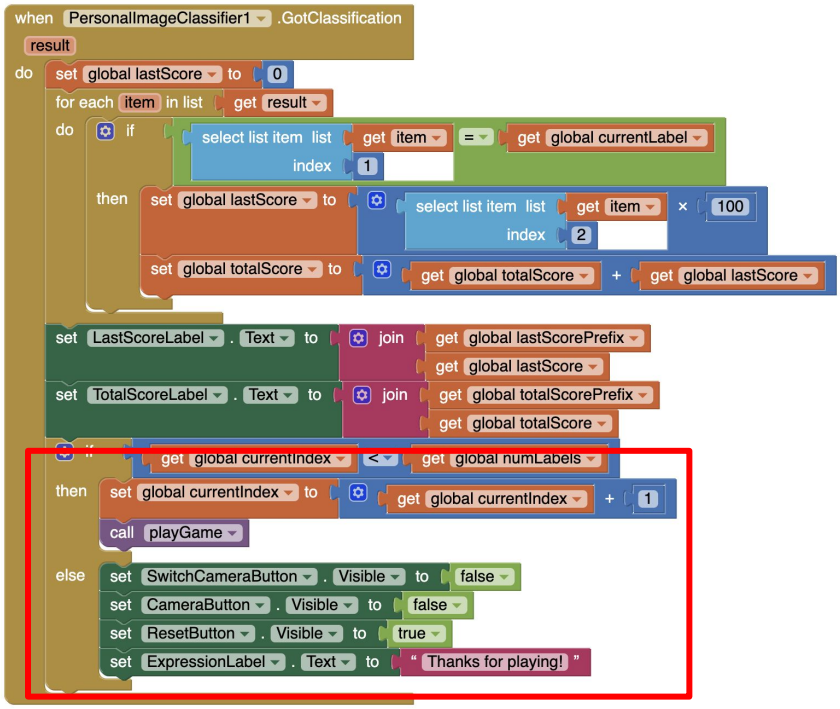


29 Now, set the LastScoreLabel and TotalScoreLabel based on the new values.



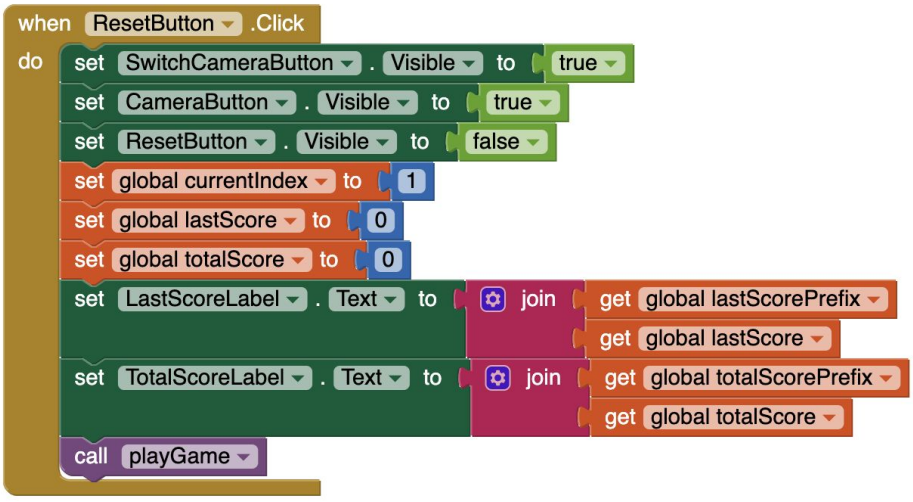
Now you can add blocks for progressing the game. Check if there are more labels (expressions) in the model. If so, continue prompting the player to take more pictures. Otherwise, end the game.

30 Add an **if then else** block. Compare **currentIndex** to **numLabels** to see if you have reached the end of the list of labels. If not, increment **currentIndex** by **1** and call **playGame** again. If you've reached the end of the list, hide the game Buttons, show the **ResetButton** so the user can play again, and display a message to let the player know the game is over. This message uses the **ExpressionLabel**.



31 Finally, add functionality to this Button. When the player clicks on it, reset the state of the game so that the player can start over.

This includes making the camera buttons visible, hiding this button, resetting the index and score variables, resetting the score text labels, and finally calling **playGame** to start the game over.



32 That's it! Try playing the game you just made and see how high of a score you can get. Also, try letting other people play your version of the game! What happens?

33 If you want to, you can also go back to the Classifier interface to train different models to use. All you need to do is upload the new models and select them in the extension's "Properties" tab.