Video #5: Adding Actions to the Change Blindness App

In this second video about our Change Blindness App, we'll examine the code that was added to implement the actions in the program. Let's first review the actions we want to perform. The user can select an image pair from this Drop Down menu, and there are four scenes to choose from: this airplane scene, a couple having dinner, a farm scene, and a marketplace. So one action we need to implement is to set up the two images for the movie, when the user makes a new selection. When we click on the run test button, a movie is shown that alternates between the two images, in a continual loop. The images are each displayed for an amount of time specified in this presentation time box, and there's a gap of time between the two images that's given in this gap time box. There's a change taking place between the two images, and as soon as the user notices the change, they click on the stop button to stop the movie. All this happens when we press the run test button. At any time, we can show the results, which displays a bar plot in the Axes area that indicates how many times we needed to repeat the pair of images before noticing the change. And when the user is done, we can click the close button to terminate the program.

Let's now have a look at the code in App Designer by selecting the Code View. At the top of the code file is a set of properties corresponding to the GUI components that we created in our visual layout, and recall that these components are stored in a structure named app, so we'll refer to them in the code as app.componentName, like app.imagesDropDown. When writing code for an App, we need to think about what data is needed for the program, and whether there's information that we need to access from multiple callback functions in the code file. We can define new properties at the top of the code file, to store this information, and these properties will be created when the App first starts up. For this App, there are two things of this sort. The first is the image pair. I created two properties, im1 and im2, to store the two images that will be

displayed as a movie, and I initially loaded the airplane images into these two properties. I also created a vector of four elements to store the results of the experiment - the number of times the images were repeated before the user noticed the change, for each of the four scenes. These new properties will also be stored in the app structure, so we'll refer to them in our code as app.im1, app.im2, and app.results.

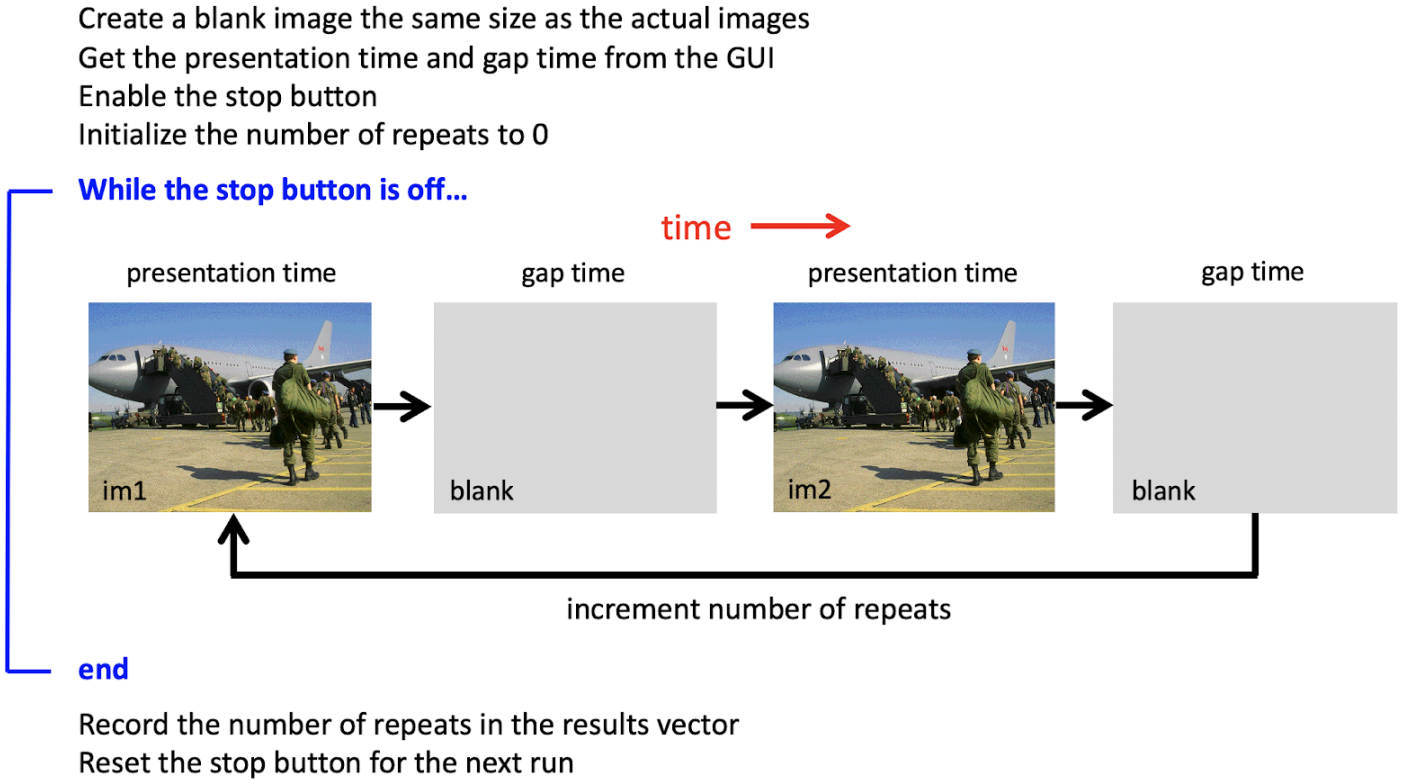
```
properties (Access = private)
    % variables to store image pair, initialized to airplane images
    im1 = imread('plane1.png');
    im2 = imread('plane2.png');
    % results of tests: number of repeats needed to detect change, for
    % each of the four pairs of images
    results = zeros(1,4);
end
```

When the user makes a new selection through the Drop Down menu, we can immediately load the images for that new selection. A Drop Down component has a valueChanged callback function associated with it - for the menu in this App, name is imagesDropDownValueChanged. MATLAB provided an initial code statement that created a local variable named value and assigned it to the Value property of the Drop Down menu. This is the particular string that the user selected, which will be airplane, dinner, farm, or market. We can compare value to these different possibilities using strcmp, and when we get a match, we'll load the corresponding image files into the app.im1 and app.im2 properties.

```
% Value changed function: imagesDropDown
function imagesDropDownValueChanged(app, event)
    % load selected image pair
    value = app.imagesDropDown.Value;
    if strcmp(value, 'airplane')
        app.im1 = imread('plane1.png');
        app.im2 = imread('plane2.png');
    elseif strcmp(value, 'dinner')
        app.im1 = imread('dinner1.png');
        app.im2 = imread('dinner2.png');
    elseif strcmp(value, 'farm')
        app.im1 = imread('farm1.png');
        app.im2 = imread('farm2.png');
    else
        app.im1 = imread('market1.png');
        app.im2 = imread('market2.png');
    end
end
```

The most extensive actions take place when we press the "run test" button - let's explore that code next. This button is named demoButton and we create a demoButtonPushed callback function that's invoked whenever the user presses this button. To understand the steps of the code, I made a slide that first describes these steps in English. The heart of the function is the

movie that displays the images im1, then im2, in a continual loop over time. The images are each displayed for an amount of time that's the presentation time that the user entered on the GUI. Then during the gap between the images, I actually display a blank image that's just a uniform gray image that's the same size as im1 and im2, and this blank image is shown for the gap time that the user entered on the GUI. While we repeat this loop over and over, we keep track of the number of times it's repeated, incrementing that number each time through the loop. Before getting into this loop, we create a blank image that's the same size as the actual images, get the presentation and gap times from the GUI, then enable the stop button so that the user can now press this button to stop the movie, and initialize the number of repeats to 0. Remember that the stop button is a State Button that flips between two states when you press it. The button starts out in an off state, and as long as it stays off, we continue the movie loop. At some point, the user will press the stop button, which changes its state to on, and the loop will stop. When the loop ends, we record the number of repeats in the result vector that was created, and reset the stop button for the next time the user presses the run test button.



So now let's connect these steps to the code. We first get the size of the actual images and create the blank image using the ones function, that's the same size. The numbers that the user specified for the presentation and gap times are in Edit Fields and they're the Value property of these fields. Then we set the Enable property of the stop button to 'on' and initialize the number of repetitions, nreps = 0. Then we have the movie loop, while the Value property for the stopButton remains 0, we display im1, wait, display the blank image, wait, im2 wait, blank again and wait. Remember that testImage is an Image GUI component, and we can display a new image there by setting its ImageSource property to the matrix storing the image. After the loop is done, we want to store the number of repeats in the results vector, but we first need to figure out

which index to use. This is the same as the index of the item that the user selected from the Drop Down menu of images. So we can use the same strategy that we used in our energyApp program to figure out which energy source the user selected. We can compare the string selected to the cell array of items in the list, and find the place where this is true - that gives us an index we can use to store the results of this run of the experiment. Finally, we reset the stop Button for the next run - set its Value back to 0 and disable it by setting its Enable property to 'off'.

```
% Button pushed function: demoButton
function demoButtonPushed(app, event)
    % create blank frame of the same size as the image pair
    [rows, cols, rgb] = size(app.im1);
    blank = 0.8*ones(rows, cols, rgb);
    % get presentation time and gap time from the app
    presentTime = app.presentEditField.Value;
    gapTime = app.gapEditField.Value;
    % enable the stop button during presentation of images
    app.stopButton.Enable = 'on';
    nreps = 0;
    % keep alternating between the two images, with blank frame
    % in between, until user clicks the stop button
    while (app.stopButton.Value == 0)
        app.testImage.ImageSource = app.im1;
        pause(presentTime);
        app.testImage.ImageSource = blank;
        pause(gapTime);
        app.testImage.ImageSource = app.im2;
        pause(presentTime);
        app.testImage.ImageSource = blank;
        pause(gapTime);
        nreps = nreps + 1;
    end
    % record results (number of repeats needed to detect change)
    index = find(strcmp(app.imagesDropDown.Items, app.imagesDropDown.Value));
    app.results(index) = nreps;
    % reset and disable the stop button
    app.stopButton.Value = 0;
    app.stopButton.Enable = 'off';
end
```

There's just a little more code here. When the "show results" button is pressed, this resultsButtonPushed function is called, which displays the results on the resultsAxes as a bar plot. And we can adjust the range of values to reflect the range of data in the results vector.

```
% Button pushed function: resultsButton
function resultsButtonPushed(app, event)
    % show number of repeats for each image pair as a bar graph
    bar(app.resultsAxes, app.results);
    app.resultsAxes.YLim = [0 max(app.results)];
end
```

Finally, we have a closeButtonPushed callback function that's called when the user presses the close button, that has the single statement delete(app).

```
% Button pushed function: closeButton
function closeButtonPushed(app, event)
    % close the app by deleting it
    delete(app)
end
```

So that's the implementation of an App to demonstrate the visual phenomenon known as Change Blindness. We used a couple new GUI components here, the Image component and numeric Edit Fields, and we introduced the idea of disabling and enabling a GUI component like a button, which allows us to control what components the user is allowed to interact with at a particular time in the running of the program. And we also saw how we can initiate a process that repeats itself continually, like our continually playing movie, and allow the user to interrupt the process. That's something that's very easy to do with a graphical user interface.