Figure Basics in MATLAB

Throughout the semester, we have used figures to display images, plots and do animation. For this project, you will learn how to manipulate figures to create what is called a Graphical User Interface, or GUI (pronounced Gooey, with a hard G) for short, so that way we can create a game of 2048! This tutorial, coupled with the ABCs_figures, is designed to give you the basic knowledge necessary to complete this project! If you want to include things in your project not covered here or you just want to learn more, the MATLAB documentation online will have all the information you need! Good Luck! Have fun! And please remember the project is supposed to give you an opportunity to learn something outside the normal bounds of this course, so you can take the information you learn here and online to do just about whatever you want!

Object Oriented Programming

Before we get into all things figures, we must first take a foray into something called Object Oriented Programming. This is a concept used in programming languages, like Java, where everything is centered around the idea of a class, which is kind of this box that you can set the characteristics of and then do operations on. In fact, we have already seen this concept of classes in MATLAB. Take *struct* for example. A structure has the characteristics of being able to store an unlimited number of fields and each field can hold any data type. We can also do operations on structures, such as use functions `getfield`, `setfield`, and `rmfield`, as well as concatenate structures together to create structure arrays. The important concept here is that these operations only work on structures. For example, you can concatenate two structures together, but you cannot concatenate a structure with a cell. The idea of classes is very powerful because classes can be used to create objects. An object is a specific instance of a class. For example, you have the class *struct*, which tells you information (i.e. its characteristics) about structures but you cannot actually perform an operation unless you create a structure. Therefore, if you create a 1x1 structure and store it in a variable `st`, then `st` would be considered a structure object, or, in other words, an instance of the *struct* class.

Another important and powerful part of the idea of classes is that classes can be extended, meaning you can create a new class that is sub class of your original class. This sub class, called a child, has the same characteristics of the original class, called the parent class, with a few more added on. The easiest way to think about this is to think of parent and child classes as dogs. All dogs (i.e. the parent class) have 4 legs, have a tail, and bark. Dalmatians (i.e. a child class) have all the same characteristics of dogs, but also have some extra ones like have black and white spots, have short ears, and live at fire stations. One important thing to note is that each parent class can have an unlimited number of child classes, but a child class can only have one parent class.

By now, you may be thinking this is super difficult and confusing and what does this have to do with figures? The answer, is that the way MATLAB deals with figures draws a lot of inspiration from object oriented programming, but is a lot easier to work with than you would think!

Figures

Now that we have talked about object oriented programming, lets get into figures! **A figure (or figure window) is a container for graphics or user interface components.** By setting the **properties** of a figure, we change how the figure displays what is has. A property is similar to a field for structures except instead of holding information, a property tells MATLAB how to display the figure or when/how to do something. Here's what we can do and how we work with figures:

I. Creating figures

Many functions like `imshow` and `plot`, automatically create figures and then fill them with something, however if we to create a blank figure we use the `figure` function.

II. Identifying figures

When we work with figures, MATLAB needs someway to identify which figure we are talking about. This identifier is called a figure handle (usually f for short). This is similar to the way MATLAB uses file handles when dealing with files. You can get the figure handle one of two ways

1. `f = gcf` **gcf** stands for "get current figure" and returns the file handle for the current figure. You want to use this one when you already have an existing figure
2. `f = figure` if you give `figure` an output, then it will create a new figure and return a figure handle for that figure.

III. Setting Properties

As stated previously, the properties of a figure tell MATLAB how to display a figure. To set or change a property, you use the dot operator. For example, if you have figure with figure handle `f` and you want to name the figure 'Neat' you would change the Name property:

```
f.Name = 'Neat'
```

You can also set properties when you create a figure by giving the `figure` function inputs. The general format is

`figure(property, value, property, value, ..., property, value)`

where the property is a string containing the name of the property and value is what you are setting that property too. For example, to create a figure named 'Neat', you would do

```
figure('Name', 'Neat')
```

For a full list of properties and what values you can set them to click here.

IV. Closing a figure

Once you are finished with a figure and want to close it in your code use

`Close(f)`

UIControl

Now that we have a figure and know how to change it around, we now need something to put inside of our figure that will allow us to display things and interact with our figure. The best

way to do so is by using a **UIControl.** A UIControl is an object that you can place inside a figure and use it to display information (text), use it to gather user inputs (check boxes, sliders, text input), or have the user control things (buttons).

I. Creating

In order to put UIControls into a figure, we must first create them. We do so by using the `uicontrol` function which used as follows:

```
h = uicontrol(parent, property, value, ... , property, value)
```

where h is the object handle (which is what you will use to reference this UIControl just like figure handles), `parent` is the figure handle of the figure you want your UIControl to go to, and `property, value` sets a property of the uicontrol to a value.

When you use `uicontrol`, you must give it an output so you have an object handle for reference. If you do not give `uicontrol` a `parent` input, it will automatically assign it to the current figure (which is ok if you only have 1 figure, but it's better to give a `parent` input).

II. Properties

Just like with figures, UIControls have different properties that define the characteristics of the UIControl. We will go over just a few of the important properties below, but for a full list of properties and their associated values, click here.

1. Style – this property defines what the UIControl actually is, such as whether is a textbox, a pushbutton, a text input, or something else. We will only cover the 3 values to create the aforementioned styles, but to find a full list, follow the link to the full list of UIControl properties above.
    a. `'text'` – creates a UIControl that only displays text.
    b. 'pushbutton' – creates a button on the screen that you can click on. We will learn later how to make clicking the button do something
    c. 'edit' – creates a text input box where the user can type something
2. String – this property defines what text your UIControl displays. For textboxes and buttons, this property just shows up on the UIControl. For text inputs, this property defines the default text in the UIControl, when a user changes what is in the user input box, the user is actually changing the String property of the text input UIControl.

At this point, you should complete the ABCs_figures to make sure you understand how to implement what was discussed here. Next, we will discuss how to make figures respond to user input

User Input

Being able to precisely control how a figure window looks and displays information is pretty neat, but the real power of figures is that we can make it so the user can interact with the window itself.

I. Callback Functions

Callback functions provide the basis for being able to perform actions from inside a figure. Take, for example, a button. As of right now, if we put a button in a figure, we can click it and nothing happens. However, if we use a callback function, we can make it to where clicking the button makes something happen.

1. Writing

Writing a callback functions is very simple. It is exactly like writing a helper function, except that the first two inputs have to be `src`, and `event`. `src` (which stands for source) has the UIControl which triggered the callback function (i.e. the button that the user pressed which made the callback function run). Event contains information about the call to the callback function. The basic function header of a callback function is as follows

```
function functionhandle(src,event,input1, input2, ...)
```

This function would perform some action, like updating information, creating more UIControls or anything else you can think of.

2. Calling

Calling callback functions can be a bit confusing because they are actually values to properties of figures and UIControls. We will discuss the properties that control callback functions in a later section, but for now, just know that a pushbutton has a property called Callback that we can use to make our callback function run when someone clicks our button.

Assuming we have a button, who's handle is `h`, then the general form of assigning a callback function is as follows:

```
h.Callback = {@functionhandle, input1, input2, . . . }
```

where the value of the Callback property is a cell array whose first cell contains the function handle and the rest of the cells contain the inputs that correspond to our inputs in our callback function header. Please note that the function handle is just the name of a function and that you do not have to include `src` and `event` in your cell array. If your callback function has no other inputs besides `src` and `event`, then you do not have to put the function handle in a cell array.

EXAMPLE:

If we have two buttons whose handles are `btn1` and `btn2` and two callback functions whose headers are as follows:

```
function myCallback1(src, event)
function myCallback2(src, event, input1, input2)
```

then,

```
btn1.Callback = @myCallBack1
btn2.Callback = {@myCallBack2, input1, input2}
```

are both valid assignments to the Callback property that will allow the functions `myCallback1` and `myCallback2` to run when a user clicks `btn1` or `btn2` respectively.

If you want more information on how to create and use callback functions, click here.

3. Interactive Control Properties

    These are some properties of figures and UIControls that allow callback functions to run after the user interacts with an object. There are a lot more that can be found in the above link

    i. Callback: figures and UIControls – for when the user triggers an object, like by pushing a button
    ii. CloseRequestFcn: figures – for when you want a function to run when a user closes the figure
    iii. KeyPressFunction: figures and UIControls – for when you want an action to happen when a user pushes a specific key on the keyboard while the pointer is over an object

II. Changing and Deleting UIControls

Being able to change and delete UIControls from within callback functions is very important. Say for instance that you have a textbox that displays a welcome message on the start screen of a game. When you start playing the game, you need to either make it invisible by changing its visibility (which is a property) or deleting it altogether.

1. Retrieving UIControls from a figure

    Since all UIControls are children of a figure, they can be thought of as being stored as part of the figure, which allows you to access them via the figure handle. MATLAB has a function that retrieves all the children of a figure (and thus the UIControls).

    ```
    Childs = allchild(f)
    ```

    Where `childs` is an Mx1 graphics array (meaning it holds graphics objects) or an Mx1 UIControl array (meaning it only holds UIControls) and `f` is a file handle. You can index out specific UIControl objects just like you would with regular arrays. It is important to note here that the children of your figure are stored as a stack, meanining the more recently you put an object in a figure, the lower its index is when you call `allchild`.

    EXAMPLE:

    If we have,

    ```
    f = figure;
    txt1 = uicontrol(f, 'Style','text');
    txt2 = uicontrol(f, 'Style','text');
    ```

    then if we call

    ```
    childs = allchild(f);
    ```

    then `txt1` will be located at index 2, because it was created first and `txt2` will be located at index 1 because it was created after `txt1`

2. Deleting UIControls

   Deleting UIControls is as simple as

   ```
   delete(h)
   ```

   Where `h` is a handle for a UIControl object.

   Something important to understand when deleting UIControls is that you change the indices of all the other children of figure, just like you change the indices of other numbers in a numerical array when you delete an index.

   EXAMPLE:

   Given the example code from section 1 above, then if we delete txt2 by saying

   ```
   delete(txt2)
   ```

   Then when we call `childs = allchild(f)`,

   `childs` will be a 1x1 UIControl array where `txt1` will be stored in the first index.

3. Changing UIControls

   This is accomplished by retrieving the UIControl objects from a figure and then changing their properties

At this point, you should complete the ABCs_Callback to make sure you understand how to implement what was discussed here.

Final Note

This guide has been written to only be an introduction into figure's in MATLAB. There is so much more that can be done than described here. For more information on the topics discussed here or on topics not covered, the MATLAB documentation is the best place to look. I hope this guide has been helpful. Happy Coding!