Project Taste: Octave Coffee Machine Simulator

Sanjin Ruzic

January 3, 2024

Contents

1 Introduction

This is a comprehensive documentation and guide for Taste, a project I have been developing in GNU Octave, tool best known for its numerical computation capabilities, for my final ENS101 (Introduction to Engineering) project assignment. The cornerstone of Taste is its friendly graphical user interface (GUI), which faithfully emulates the principle of work and characteristics of coffee vending machines in the real world. Not only does it enable you to select from a wide variety of coffee types, but it also offers the customization of your selection to some degree (it is worth noting that it is purely cosmetic and does not affect the brewing process in any way). An added benefit is that you are free to tailor your own experience according to your wishes by specifying the strength of the coffee, quantity of sugar and even simulate actions such as inserting coins into the machine slot, taking change and collecting dispensed coffee.

In addition to replicating a realistic physical interaction scenario, Taste utilizes underlying logic fairly identical to coffee machines in the real world, which can be observed with how it handles crucial resources such as the water level, amount of coffee beans and machine money balance, alerting the user in the event of any one's depletion unless refilled by the operator. A deeper examination may lead to the acknowledgement that my project intends to explore a more practical application of key engineering principles taught during the ENS101 course.

The aim of this documentation is to provide a detailed overview of the design features of the project, explaining the UI components it consists of and their usefulness in a broader sense. I will also be discussing the steps I have taken in the GUI creation so that it may serve as a valuable insight into the development process and capture the full picture of my approach. Lastly, it will delve deeper into the intricacies of the functional logic responsible for the coffee machine's key operations and interactions with the user.

2 Usage Manual

Using the Taste coffee machine GUI is fairly straightforward. Here are the steps:

1. **Start the Program:** Open the file `coffeeVendingMachine.m` in GNU Octave and run the script. After you have done this you should see the GUI window appear on your screen.

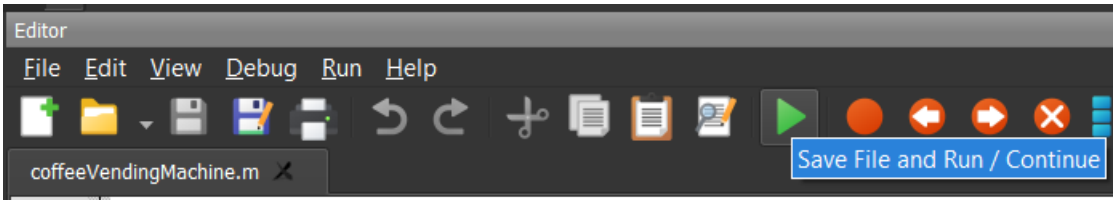

Figure 2.1 Running the script file in GNU Octave

2. **Select Coffee:** On the GUI, you will be presented with several options for coffee types. Click on the `Select` on the panel of the coffee type that you want.

Figure 2.2 Selecting a coffee

3. **Customize Your Coffee:** Once you have made your selection, a mini window will appear displaying more details about the coffee you have chosen. You may also customize your coffee according to your preference by selecting the coffee strength and sugar level buttons, one from each group at most.

Figure 2.3 Coffee customization options

4. **Insert Money:** You may view the price of the coffee in the same coffee details window, not only on the main panel. Hover over the `Actions` menu bar and insert the required amount with the options provided (euros are the only available currency). The GUI will track the total amount of money you inserted and be updated accordingly for each interaction.

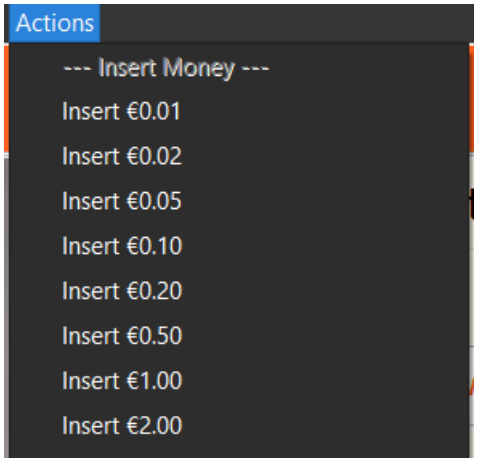

Figure 2.4 Inserting coins into the machine

5. **Collect Change:** If you inserted more money than the coffee cost, there will be change that you have to collect before the machine can begin brewing your coffee, which is deducted from the machine's money balance. Even if the change is zero, you must press `Collect Change` in the `Actions` menu bar in order for the machine to proceed to the brewing stage.

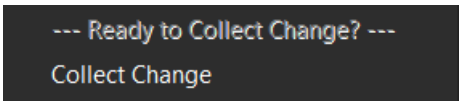

Figure 2.5 Taking the change

6. **Collect Coffee:** Be patient while the machine is making your coffee. The prompt will inform you when it is finished and instruct you to collect the dispensed coffee. To do this, simply go to the `Actions` menu bar and press `Collect Coffee`. And just like that, you are done! The mini window automatically closes after a few seconds of inactivity and you may again repeat the process of ordering coffee by following these steps.

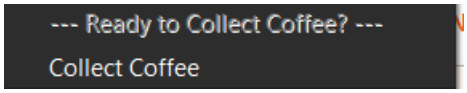

Figure 2.6 Collecting the coffee

3 Design Overview

In this section, I will review the architectural aspects of the GUI. This includes GUI initialization, creation and organization of UI panels and elements, as well as the addition of a menu bar for user actions.

3.1 Initializing the GUI

```
% Get screen size and calculate figure position
screenSize = get(0, 'screensize');
figureSize = [1000, 700];
mainFramePos = [(screenSize(3) - figureSize(1))/2, (screenSize(4)
   - figureSize(2))/2, figureSize(1), figureSize(2)];

% Create GUI frames
mainFrame = figure('name', 'Coffee Vending Machine Simulation',
'NumberTitle', 'off',
'toolbar', 'none',
'resize', 'off',
'position', mainFramePos,
'visible', 'on');
mainFrame = gcf;
titleFrame = uipanel('parent', mainFrame,
'backgroundcolor', titleBackgroundColor,
'position', [0 0.9 1 0.1]);
coffeeSelectionFrame = uipanel('parent', mainFrame,
'backgroundcolor', backgroundColor,
'position', [0 0 1 0.9]);

% Text elements for title and clock
titleText = uicontrol('parent', titleFrame,
'units', 'normalized',
'style', 'text',
'string', 'taste',
'fontsize', 23,
'fontname', 'Georgia',
'backgroundcolor', titleBackgroundColor,
'foregroundcolor', titleTextColor,
'position', [0.25 0 0.5 1],
'horizontalalignment', 'center',
'fontweight', 'bold');
clockText = uicontrol('parent', titleFrame,
'units', 'normalized',
'style', 'text',
'string', datestr(now, 'HH:MM'),
'fontsize', 16,
'fontname', 'Calibri',
'backgroundcolor', titleBackgroundColor,
'foregroundcolor', titleTextColor,
'position', [0.8 0 0.25 1],
'horizontalalignment', 'center',
'fontweight', 'bold');
```

The purpose of the code block above is to initialize the GUI for the coffee machine simulator. First, it retrieves the screen size and defines the figure to be 1000 pixels wide and 700 pixels tall so that it may calculate the position of the main frame (GUI window), placing it at the center of the screen.

By using the **figure** function, I am able to create a new figure window with the properties specified, meaning it will be visible, but will not have a toolbar or a number title, nor will it be resizable. Two panels, `titleFrame` and `coffeeSelectionFrame` are then created within the main frame with the **uipanel** function, which, similar to before, generate two panel objects parented to the main frame with the given properties.

Finally, I create text elements parented to the title frame with the **uicontrol** function so that I may incorporate interactive controls, in this case two static controls that only display text in a string, namely the string `"taste"` and current time in the `HH:MM` format.

3.2 Setting up the coffee selection panel

```
% Spacing and panel dimensions for coffee panels
spacing = 0.03;
panelWidth = (1 - 6 * spacing)/5;
panelHeight = (1 - 5 * spacing)/2;

% Create and arrange coffee panels
for i = 1 : length(machineData.coffeeTypes)
row = floor((i - 1)/5);
col = mod(i - 1, 5);
panelX = spacing + col * (panelWidth + spacing);
panelY = 1 - spacing - (row + 1) * (panelHeight + spacing);
coffeePanel = uipanel('parent', coffeeSelectionFrame,
'backgroundcolor', 'white',
'position', [panelX panelY panelWidth panelHeight],
'bordertype', 'etchedin');

% Load coffee image directly and display name and price
coffeeImage = imread(strcat('./CoffeeImages/', machineData.
   coffeeTypes{i}, '.png'));
uicontrol('parent', coffeePanel,
'style',
'pushbutton',
'cdata', coffeeImage,
'units', 'normalized',
'position', [0 0.3 1 0.7]);
uicontrol('parent', coffeePanel,
'style', 'text',
'string', machineData.coffeeTypes{i},
'fontsize', 12,
'fontname', 'Calibri',
'backgroundcolor', 'white',
'foregroundcolor', coffeeTextColor,
'units', 'normalized',
'position', [0.05 0.2 0.8 0.1],
'horizontalalignment', 'left',
'fontweight', 'bold');
uicontrol('parent', coffeePanel,
'style', 'text',
'string', [' ' sprintf('%.2f', machineData.coffeePrices(i))],
```

```
'fontsize', 12,
'fontname', 'Calibri',
'backgroundcolor', 'white',
'foregroundcolor', '#FF5F1F',
'units', 'normalized',
'position', [0.15 0.05 0.8 0.1],
'horizontalalignment', 'right',
'fontweight', 'bold');
uicontrol('parent', coffeePanel,
'style', 'pushbutton',
'string', 'Select',
'units', 'normalized',
'fontsize', 10,
'fontname', 'Calibri',
'position', [0.05 0.1 0.5 0.1],
'callback', {@coffeePanelCallback, i, mainFrame});
end
```

The first portion of the code sets the spacing between the coffee panels, and calculates their width and height dimensions. This is done by subtracting the total spacing from all of the available space (this is 1), and dividing this result by the number of panels.

The for loop iterates over each coffee type stored in `machineData.coffeeTypes`, and there are 10 coffee types, so it will iterate 10 times. The row and column of each panel are calculated based on the current iteration and subsequently the first two parameters of the position vector, `panelX` and `panelY`, are also calculated to determine the position where the panel will be placed on the GUI. Since there are 10 types in total, the code will dynamically generate the grid, so here it will create a 2x5 layout of panels - first five types make up the first row and the remaining five the second. For example, the row and column index for the 7th coffee type is given by:

$$\text{row} = \left\lfloor \frac{7-1}{5} \right\rfloor = \left\lfloor \frac{6}{5} \right\rfloor = 1$$
$$\text{col} = (7-1) \mod 5 = 1$$

So the 7th coffee type will be placed in the second row and second column, since both are zero-indexed.

A new uipanel named `coffeePanel` is created as child of `coffeeSelectionFrame` for each coffee type, it has a white background, `etchedin` border type, and its position and size are set by the previously calculated variables. I specifically created a pushbutton control to allow for on-demand image loading by calling the `loadImageCallback` function which loads the image for the selected coffee type. Then the image file from the `./Coffee Images` directory is read using **imread** function coupled with the **strcat** function that will concatenate the file extension and file name for every type of coffee to form a full file path.

The first pushbutton creates a uicontrol with the `cdata` property to ensure that the image data is properly set, thereby displaying the coffee image as an icon. The second and third uicontrol are responsible for text labels for coffee name and price, while the fourth creates a `Select` pushbutton that when clicked invokes the `CoffeePanelCallback` function.

3.3 Setting up menubar

```
function coffeePanelCallback(hObject, eventdata, coffeeIndex,
    mainFrame)
% Create mini window panel
miniWindow = uipanel('parent', mainFrame, 'backgroundcolor',
    backgroundColor, 'position', [0.1 0.1 0.8 0.8]);

% Actions menubar
actions = uimenu('label', 'Actions');
insertMoneyHeading = uimenu(actions,
'label', '--- Insert Money ---',
'enable', 'off');
insert1Cent = uimenu(actions, 'label', 'Insert 0.01', 'callback',
    {@insertMoneyCallback, 0.01, coffeeIndex, mainFrame, miniWindow
    });
insert2Cents = uimenu(actions, 'label', 'Insert 0.02', 'callback',
     {@insertMoneyCallback, 0.02, coffeeIndex, mainFrame,
    miniWindow});
insert5Cents = uimenu(actions, 'label', 'Insert 0.05', 'callback',
     {@insertMoneyCallback, 0.05, coffeeIndex, mainFrame,
    miniWindow});
insert10Cents = uimenu(actions, 'label', 'Insert 0.10', 'callback'
    , {@insertMoneyCallback, 0.10, coffeeIndex, mainFrame,
    miniWindow});
insert20Cents = uimenu(actions, 'label', 'Insert 0.20', 'callback'
    , {@insertMoneyCallback, 0.20, coffeeIndex, mainFrame,
    miniWindow});
insert50Cents = uimenu(actions, 'label', 'Insert 0.50', 'callback'
    , {@insertMoneyCallback, 0.50, coffeeIndex, mainFrame,
    miniWindow});
insert1Euro = uimenu(actions, 'label', 'Insert 1.00', 'callback',
    {@insertMoneyCallback, 1.00, coffeeIndex, mainFrame, miniWindow
    });
insert2Euros = uimenu(actions, 'label', 'Insert 2.00', 'callback',
     {@insertMoneyCallback, 2.00, coffeeIndex, mainFrame,
    miniWindow});
takeChangeHeading = uimenu(actions,
'label', '--- Ready to Collect Change? ---',
'enable', 'off');
collectChange = uimenu(actions,
'label', 'Collect Change',
'callback', {@brewAndCollectChange, coffeeIndex, mainFrame,
    miniWindow});
takeCoffeeHeading = uimenu(actions,
'label', '--- Ready to Collect Coffee? ---',
'enable', 'off');
collectCoffee = uimenu(actions,
'label', 'Collect Coffee',
'callback', {@collectCoffeeCallback, mainFrame, miniWindow,
    actions});
```

This code first creates a uipanel parented to the main frame whose background color is set to the pre-defined variable and sets its position. The subsequent lines of code initialize a menubar using the **uimenu** function with the label `Actions`, separating it into UI menus for headings and interactions with the machine that invoke the `insertMoneyCallback`, `brewAndCollectChange` and `collectCoffeeCallback` functions respectively. These callback functions take in the following arguments: function handle, amount, coffee index, main frame and mini window.

3.4 Designing the coffee details mini window

```
closeButton = uicontrol('parent', miniWindow,
'style', 'pushbutton',
'units', 'normalized',
'position', [0.9 0.9 0.1 0.1],
'string', 'X',
'foregroundcolor', 'white',
'backgroundcolor', 'red',
'fontsize', 16,
'fontname', 'Calibri',
'fontweight', 'bold',
'callback', {@closeMiniWindow, miniWindow, actions});

% Display coffee image on the left of mini window
pkg load image;
selectedCoffeeImage = imread(strcat('./CoffeeImages/', machineData
   .coffeeTypes{coffeeIndex}, '.png'));
coffeeImageAxes = axes('parent', miniWindow,
'units', 'normalized',
'position', [0 0 0.3 1],
'visible', 'off',
'color', backgroundColor);
coffeeImageAxesUnits = get(coffeeImageAxes, 'units');
set(coffeeImageAxes, 'units' , 'pixels');
coffeeImageAxesPos = get(coffeeImageAxes, 'position');
set(coffeeImageAxes, 'units' , coffeeImageAxesUnits);
resizedCoffeeImage = imresize(selectedCoffeeImage, [
   coffeeImageAxesPos(4) coffeeImageAxesPos(3)]);
imshow(resizedCoffeeImage, 'parent', coffeeImageAxes);

% Display coffee info
coffeeTypeText = uicontrol('parent', miniWindow,
'style', 'text',
'string', machineData.coffeeTypes{coffeeIndex},
'units', 'normalized',
'position', [0.35 0.9 0.3 0.1],
'backgroundcolor', backgroundColor,
'foregroundcolor', coffeeTextColor,
'fontsize', 20,
'fontname', 'Calibri',
'horizontalalignment', 'left');
coffeePriceText = uicontrol('parent', miniWindow,
```

```
40 'style', 'text',
41 'string', [' ' sprintf('%.2f', machineData.coffeePrices(
       coffeeIndex))],
42 'fontsize', 14,
43 'fontname', 'Calibri',
44 'backgroundcolor', backgroundColor,
45 'foregroundcolor', '#FF5F1F',
46 'units', 'normalized',
47 'position', [0.6 0.9 0.1 0.1],
48 'horizontalalignment', 'right',
49 'fontweight', 'bold');
50 dividerAxes = axes('parent', miniWindow,
51 'units', 'normalized',
52 'position', [0.23 0.4 0.85 1],
53 'visible', 'off');
54 dividerLine = line('parent', dividerAxes,
55 'xdata', [0.35 0.85],
56 'ydata', [0.85 0.85],
57 'color', 'k');
58
59 % Add buttons for coffee strength and sugar level
60 uicontrol('parent', miniWindow, 'style', 'text', 'string', 'Coffee
       Strength', 'units', 'normalized', 'position', [0.35 0.8 0.3
       0.05], 'backgroundcolor', backgroundColor, 'foregroundcolor', '
       black', 'fontsize', 14, 'fontname', 'Calibri');
61 for i = 1 : length(machineData.coffeeStrengths)
62 uicontrol('parent', miniWindow, 'style', 'togglebutton', 'tag', '
       coffeeStrength', 'string', machineData.coffeeStrengths{i}, '
       units', 'normalized', 'position', [0.35+(i-1)*0.1 0.7 0.09
       0.09], 'backgroundcolor', 'white', 'foregroundcolor', '#FF5F1F'
       , 'fontsize', 10, 'fontname', 'Calibri', 'callback', {@
       updateButton}, 'fontweight', 'bold');
63 end
64 uicontrol('parent', miniWindow, 'style', 'text', 'string', 'Sugar
       Level', 'units', 'normalized', 'position', [0.35 0.6 0.3 0.08],
       'backgroundcolor', backgroundColor, 'foregroundcolor', 'black'
       , 'fontsize', 14, 'fontname', 'Calibri');
65 for i = 1 : length(machineData.sugarLevels)
66 uicontrol('parent', miniWindow, 'style', 'togglebutton', 'tag', '
       sugarLevel', 'string', machineData.sugarLevels{i}, 'units', '
       normalized', 'position', [0.35+(i-1)*0.1 0.5 0.09 0.1], '
       backgroundcolor', 'white', 'foregroundcolor', '#FF5F1F', '
       fontsize', 10, 'fontname', 'Calibri', 'callback', {@
       updateButton}, 'fontweight', 'bold');
67 end
```

The first section of the code block creates a close button at the top right corner of the mini window. Upon the user clicking it it will trigger the `closeMiniWindow` function. One can see that the procedure being implemented starting from line 14 is loading the image package and reading an image file corresponding to the coffee index, and ultimately resizing our image to fit the axes at full height to the left of the window. The **axes** function creates an axis object

at the position vector coordinates that is not visible and has the same color as the window's background. From line 28 onwards, the coffee information is displayed along with a divider line drawn, and below it are buttons and their labels for coffee strength and sugar level.

4 Functional Specifications

What follows is a thorough examination of the functional logic of Taste in an accessible manner. The functionalities in question refer to saving the GUI data structure, dynamic clock updates, financial transactions, handling of file input/output operations, button state management and audio playback.

4.1 GUI data handling

```
% Retrieve and save GUI data structure
machineData = guidata(mainFrame);
guidata(mainFrame, machineData);
```

The first line is retrieving the GUI data structure associated with the handle `mainFrame` and storing it in a variable called `machineData`. In the second line, the **guidata** function saves this information back to the aforementioned handle. It is essential to have this piece of code written at the beginning and end of most functions in the script; the method used clearly contributes to easier accessibility of structure members where necessary due to the function-based scope.

4.2 Dynamic clock text update

```
% Update clock text every second
while ishandle(mainFrame) && ishandle(clockText)
set(clockText, 'string', datestr(now, 'HH:MM'));
pause(1);
end
```

This while loop checks if `clockText` and `mainFrame` exist as valid handles, if yes, it updates the string property of `clockText` to the current time in the format of hours and minutes, which we obtained with the **now** function. Pausing the program lets the loop update the time displayed every second.

4.3 Handling financial transactions

```
function insertMoneyCallback(hObject, eventdata, amount,
    coffeeIndex, mainFrame, miniWindow)

if ~isfield(machineData, 'totalInserted')
machineData.totalInserted = 0;
end

```

```
global machineBalance;

% Track how much money is inserted
machineData.totalInserted = machineData.totalInserted + amount;
machineBalance = machineBalance + amount;

% Calculate change
remainingAmount = machineData.coffeePrices(coffeeIndex) -
    machineData.totalInserted;
if remainingAmount > 0
set(machineData.userPrompt, 'string', sprintf('Please insert %.2f
    more.', abs(remainingAmount)));
machineData.coffeePaid = 0;
else
change = abs(remainingAmount);

% Check if there's enough balance for change
if machineBalance < change
set(machineData.userPrompt, 'string', 'Insufficient machine
    balance to provide change.');
else
machineBalance = machineBalance - change;
set(machineData.userPrompt, 'string', sprintf('Brewing %s. Change:
     %.2f.', machineData.coffeeTypes{coffeeIndex}, change));
set(machineData.collectChange, 'enable', 'on');
machineData.coffeePaid = 1;
machineData.hasChange = 1;
```

First we check for the existence of the structure member `machineData.totalInserted` and initialize it to 0 if it does not exist, and we also access the global variable `machineBalance`. The amount inserted on callback is added to the `machineBalance` while the change is deducted. To account for the coins inserted into the machine, we will track the cumulative amount in `machineData.totalInserted`. If there is enough money in the machine, the change is calculated as an absolute value of the `remainingAmount` variable that we computed previously, and if the machine lacks the money to give change, the prompt will display an error message. The state of the boolean structure members `machineData.coffeePaid` and `machineData.hasChange` changes to 1 as preparation for the next stage.

4.4 Handling machine inventory file operations

```
% Declare global variables for machine stats
global totalCoffeeMade;
global machineBalance;
global waterLevel;
global coffeeBeans;

% Load machine inventory file data
scriptPath = mfilename('fullpath');
scriptDir = fileparts(scriptPath);
```

```
filePath = fullfile(scriptDir, 'm_inv.txt');

if exist(filePath, 'file')
fid = fopen(filePath, 'r');
fileData = textscan(fid, '%s', 'delimiter', '\n');
fclose(fid);

if isempty(fileData{1})
totalCoffeeMade = 0;
machineBalance = 100;
waterLevel = 5000;
coffeeBeans = 1000;

else
textLine1 = fileData{1}{1};
textLine2 = fileData{1}{2};
textLine3 = fileData{1}{3};
textLine4 = fileData{1}{4};

coffeeMadeStartIndex = strfind(textLine1, 'Coffee Made: ') + 13;
totalCoffeeMade = str2num(textLine1(coffeeMadeStartIndex:end));

balanceStartIndex = strfind(textLine2, 'Balance: ') + 9;
machineBalance = str2num(textLine2(balanceStartIndex:end));

waterLevelStartIndex = strfind(textLine3, 'Water Level: ') + 13;
waterLevel = str2num(textLine3(waterLevelStartIndex:end));

coffeeBeansStartIndex = strfind(textLine4, 'Coffee Beans: ') + 14;
coffeeBeans = str2num(textLine4(coffeeBeansStartIndex:end));
end
end

function collectCoffeeCallback(hObject, eventdata, mainFrame,
   miniWindow, actions)
machineData = guidata(mainFrame);

global totalCoffeeMade;
global machineBalance;
global waterLevel;
global coffeeBeans;

set(machineData.userPrompt, 'string', sprintf('Thank you! Aborting
    in 10 seconds...'));
set(machineData.collectCoffee, 'enable', 'off');

% Updates inventory and balance
scriptPath = mfilename('fullpath');
scriptDir = fileparts(scriptPath);
filePath = fullfile(scriptDir, 'm_inv.txt');

```

```
fid = fopen(filePath, 'w+');
totalCoffeeMade = totalCoffeeMade + 1;
fprintf(fid, 'Total Coffee Made: %d\n', totalCoffeeMade);
fprintf(fid, 'Machine Balance: %d\n', machineBalance);
waterUsage = 200;
coffeeUsage = 15;
waterLevel = waterLevel - waterUsage;
coffeeBeans = coffeeBeans - coffeeUsage;
fprintf(fid, 'Water Level: %d\n', waterLevel);
fprintf(fid, 'Coffee Beans: %d\n', coffeeBeans);
fclose(fid);
```

We declare four global variables for all of the machine statistics contained in the file that we will be using for the purpose of reading/writing operations. However, the script initializes preset values to the variables if the file is empty or not found. To get the full relative path of this file, several functions come into play:

- **mfilename** to return the name of the currently running script file;
- **fileparts** to return the name, directory and extension components of the parsed filename string;
- **fullfile** to form the full file path string by concatenating the directory path and filename strings.

If the file does exist, it is opened for reading and its binary data contents are read into `fileData` **fopen** function set to read mode before closure with the **fclose** function (by referencing every individual text line to be read into a separate variable). The code finds the start of the information it needs to read by finding the position of the label and adding their lengths. Lastly, when the `collectCoffeeCallback` function is invoked, the machine inventory and balance is updated and new information written back to the file.

4.5 Managing button appearance and state

```
% Manages the visual appearance and selection state of buttons
function updateButton(hObject, ~, buttonIndex)
buttonGroup = get(hObject, 'tag');
allButtons = findobj('style', 'togglebutton');
buttonsInGroup = allButtons(strcmp(get(allButtons, 'tag'),
    buttonGroup));

set(buttonsInGroup, 'value', 0, 'backgroundcolor', 'white', '
    foregroundcolor', '#FF5F1F');
set(hObject, 'value', 1, 'backgroundcolor', '#FF5F1F', '
    foregroundcolor', 'white');

selectedButton = findobj(buttonsInGroup, 'value', 1);
if ~isempty(selectedButton) && selectedButton ~= hObject
set(selectedButton, 'value', 0, 'backgroundcolor', 'white', '
    foregroundcolor', '#FF5F1F');
end
end
```

With this function, we are able to get the tag property of the button that the user pressed that called it, which is advantageous for knowing the group the button belongs to. In the code that follows we find all objects in the current figure with a "togglebutton" style, find all buttons that have this tag, and deselect all buttons in the group while setting the value of the clicked button to 1. Lines 10-12 find all button objects in the group that are selected, and check to ensure that if another button in a group is selected, all other buttons in the group that are currently selected would be deselected. The background color changes to orange if selected and white if deselected.

4.6 Audio playback

```
1  % Playing brewing sound
2  function playBrewingAudio()
3  persistent player;
4  if isempty(player) || ~isplaying(player)
5  [y, fs] = audioread('./Sounds/01_Brewing.wav');
6  player = audioplayer(y, fs);
7  play(player);
8  end
9  end
10
11 function playCoinSlotAudio()
12 persistent player;
13 if isempty(player) || ~isplaying(player)
14 [y, fs] = audioread('./Sounds/02_Coin.wav');
15 player = audioplayer(y, fs);
16 play(player);
17 end
18 end
```

The function defined here play a specific audio file when called, and checks if player is empty or not playing any audio currently. If either is true it reads the audio file from the path, creates a new audioplayer object with audio data `y` and sampling rate `fs` before playing the audio.

5 Conclusion

Using GNU Octave, I developed Project Taste according to the given instructions and criteria, making sure that I went through all the points of preparation and that no aspect was overlooked. During the process of creating my project, I learned a lot and I look forward to further expanding my level of knowledge and future work.

GitHub Repository: `https://github.com/SanjinRuzic/OctaveCoffeeMachine`

6 References

1. Octave Documentation: Figure
2. Octave Documentation: Uipanel
3. Octave Documentation: Uicontrol
4. Octave Documentation: Uimenu
5. Octave Documentation: Axis Configuration
6. Opening and Closing Files in Octave
7. Octave Documentation: Mfilename
8. Octave Documentation: Fileparts
9. Octave Documentation: Fullfile