MATLAB App Designer

This program is implemented using MATLAB's App Designer. Double-clicking the 'circles.mlapp' file will run the application. Right-clicking the file and selecting 'Edit' will open the MATLAB code in App Designer. Opening up the file from the MATLAB editor will also open the code. All the necessary information about the MATLAB App Designer can be found here:

https://www.mathworks.com/products/matlab/app-designer.html

Creating a Setup

There are a few ways to define the circles that will be drawn and the triggers that will be sent.

The first is to use the UI table directly. In the MATLAB app all of the values in the table can be changed dynamically. The 'Save' button can also be used to save the current set of data in the table to the specified csv file. The 'Load' button can also be used to load any data that has been saved from the specified csv files. When the app is closed the current data will be saved to the 'lastCircles.csv' file, and when it is reopened data will be loaded from the save file. There is no easy way to add or remove rows from the table using the MATLAB UI so it may be easier to define data in the csv files directly.

File saving and loading are handled by the 'cell2csv.m' and 'csv2cell.m' files respectively. These files were taken directly from users on the MathWorks forums.

Data can also be generated within the Generation panel of the UI. The user can define a specific RNG seed and a length of time for which the entire run of the experiment will last. The number of circles of each color to be generated can be defined, and a trigger value is associated with each of these colors. Currently, the circles are generated completely randomly and tend to overlap each other, or be unevenly spaced apart. This is something that can be improved but may not be necessary as all the required data can be manually defined. The random generation is defined in the 'randomCircles' function.

Timer

When the 'Start' button is pressed, a MATLAB timer is started. This timer will tick based on the value defined by app.tStepLength, currently set to 0.01 seconds. At each tick of the timer the 'timerFunc' function is called. This function is used to update the current timestep and continuously draw circles, play sounds, and/or send triggers.

One thing to note is the use of a FixedRate timer. This means that each tick of timer will occur at equal intervals despite the time it takes for processing. This is necessary to ensure that there is accurate timing. However, using a fixed rate means that any processing done at each tick must be faster than the length of a timestep. If it is slower, some of the steps will be skipped and not all the specified circles will be displayed.

Sound and Trigger Functions

The sound and trigger functions can be used independently or together, while simultaneously drawing circles or not. This allows the functionality to play sounds while circles are appearing, or sending triggers while circles are appearing or both. Sounds can also be played without the circles and triggers can be sent alongside the sounds. For both sound and trigger functions a single parameter is used. These parameters are defined in the Trigger column of the main UI table. When the timer starts, the sound or trigger functions will be called with the specified parameters at the timesteps defined in the StartTime column.

Sound is controlled by the function 'soundFunc'. The current implementation is to play a single tone at a frequency of 200 times the given parameter for the length specified by the SoundLength entry box. This function can be changed to include the use of sound clips or any other type functionality.

Triggers are controlled by the function 'trigFunc'. There is currently no implementation for this and the program has not yet been tested with the TriggerBox. The idea behind this would be to send a trigger at at the specified parameter at the given timestep. This can be used to indicate when a circle of a first appeared or when a sound was played.

Circle Display Function

This section explains the 'updateCircles' function within the MATLAB code. This function is called at each tick of the timer when the timer is running.

The main functionality in the program is to display a sequence of circles on screen at specified times with specified sizes. This is done with the built-in Matlab function 'insertShape' within the Image Processing Toolbox. A static black image is loaded and displayed, then filled circles are inserted into the image using this function. The one line of code which performs this action is:

```
J = insertShape(J,'FilledCircle',circleToDraw,
                'Color',colors,'Opacity', opacityNum);
```

Each circle inserted using insertShape requires the following parameters: geometry, color, and opacity value. In this case the circleToDraw parameter specifies the geometry. This is represented as a cell array of 1x3 matrices. The values in each matrix are [x, y, radius], giving the xy coordinates of the center of each circle and each circle's radius. The colors parameter above gives the color of each circle, given as a cell array of strings. For example the following code will display a red circle with a centre at (100,150) with a radius of 50 and white circle with a center at (250,200) with a radius of 75.

```
circle1 = [100 150 50];
circle2 = [250 200 75];

J = insertShape( J,'FilledCircle', {circle1 , circle2}
                 'Color',{'red', 'white'},'Opacity', 1);
```

Opacity is given as number on a scale of 0 to 1, with 1 being the maximum possible opacity. Different opacity levels are used to create the effect of a circle fading and fading out to make the appearance of circles smooth. The number of opacity levels is set by the app.opacityLevels property. This number needs to be odd, and increasing the number will add more opacity levels creating a smoother fade in and fade out. However, each opacity level requires a separate call to the insertShape function which can significantly slow down the processing, and cause timing issues where circles will not be drawn properly. An app.opacityLevels value of 9 seems to create a smooth fade in/out without causing any noticeable problems.

Circles to be drawn are represented by a single data structure, referred to as app.circleArray. This is a 3-dimensional cell array. The first dimension represents the possible time steps, the second dimension represents the possible opacity levels. The array will then have two entries for every single possible time step and every single possible opacity level. The first entry is a 1-dimensional cell array containing the geometries of all of the circles to be drawn for each timestep and each opacity level. The second entry is a 1-dimensional cell array containing the colors to be used for each of those circles. This array can be accessed to retrieve all the required information for drawing circles at any given timestep and opacity level.

The app.circleArray property is updated whenever any data in the main UI table is edited, the CircleLife parameter is changed or the Start button is pressed. It is updated using the 'circThroughTime' function which creates the array and populates it using the information given in the main UI Table and the current value of CircleLife. The main UI table defines a start time for each circle. This is the first timestep where each circle will appear. The circle will then stay for the duration defined by the value of CircleLife. For consecutive timesteps the value of opacity will change creating the fade in/out effect until the end of the circle's life.

The updateCircles function also takes two parameters: a set of axes and a timestep value. The axes are where the new image with circles inserted will be displayed. This can be any set of axes, which means the function can be called using a set of axes in a different window. The timestep value is used to extract the circles from the 'app.circleArray' data structure. At each timestep, all possible opacity levels are looped through and the data is extracted from app.circleArray to be inserted into the displayed image.