CMPT 371 – TEAM 3 DESIGN DOCUMENT

Virtual Reality Medical Imaging Software with Luxsonic Technologies Inc.

JANUARY 30, 2017

1 TABLE OF CONTENTS

1 PURPOSE

The purpose of this design document is to present our architecture's description, plan the classes and their interactions that which will be implemented, and state any changes that may occur to our design or classes as the project progresses. The architecture section will state why we chose to implement our architecture, the advantages and disadvantages of its implementation, and why chose it over other architectures that we considered. In the Unified-Modeling-Language section, we will describe what each class should do, what information they will contain, and what information they will send to other classes. It will not contain any code specifications, just how the different classes are connected. Unified-Modeling-Language diagrams are provided with detailed descriptions. This document will describe the classes that will later be implemented, and show their interactions with each other.

2 DEFINITIONS AND ACRONYMS

Digital Imaging and Communication in Medicine (DICOM): This is the primary file format used to store a series of medical images such as x-rays, ultrasounds, MRIs, and other images used in medicine.

Model-View-Controller (MVC): An architecture design, which implements the idea that classes that interact with the user (view), will send information to the controller that manipulates the information set of classes (model).

Graphical User Interface (GUI): This is the visual depiction of an interface. This interface is one that the user will be able to see, interact, and affect.

3 ARCHITECTURE DESCRIPTION

The design style used for this project will be independent components. This architecture breaks everything down into functional components that are used to create well-defined communication interfaces that contain different methods, properties and events. This allows for greater abstraction without having as much concern placed on rigid communication protocols. The various events executed will involve the use of only one or two classes without the need for other classes to be involved or affected. This will mean dependencies for each class are reduced and that each class will perform a very specific set of methods. Having this as our design choice will give us a significant advantage moving forward with the project. These advantages are described in the following section.

4 ARCHITECTURE JUSTIFICATION

Having an independent components based architecture is important for a number of reasons. The primary reason for using this architecture is that it is the easiest to implement for the software we are using. Since the Unity environment relies heavily on object creation, everything created in Unity is an object. All objects in Unity have their default objects such as Transforms (positions) attached to them. Because these objects are all independent to each other, it would be difficult to use an architecture with a rigid structure. In relation to the Model-View-Controller (MVC), having one class that acts as a controller would be difficult to implement. In addition, we fully expect to create additional functions and potential classes not currently shown in the UML diagram. Since the technology and hardware is new to many of us, changes during implementation will occur. Ideally, we would want to minimize any of these changes, but having an independent components architecture gives us the flexibility to make changes more easily during implementation.

Another significant advantage of using independent components is reusability. Since several of the classes are designed for a very specific task, they can easily be reused. Independence of classes allows them to be modified, removed, and added with minimal impact to the rest of the system. This reduces their dependencies and increases decoupling. The reduction in dependencies allows us to work on the project more effectively. We can work separately on different scripts without requiring other scripts to be completed. There will also be less concern that manipulation of different scripts would have unforeseen effects on other scripts.

5 UML DIAGRAM

The following section shows the UML diagram for all classes used in the program. Currently, only features we wish to try implementing in our first deliverable will be shown. Additional functions and classes will be implemented in future deliverables. The UML will visually show the interaction and relationships between all classes implemented. To reduce confusion, only script objects are shown.

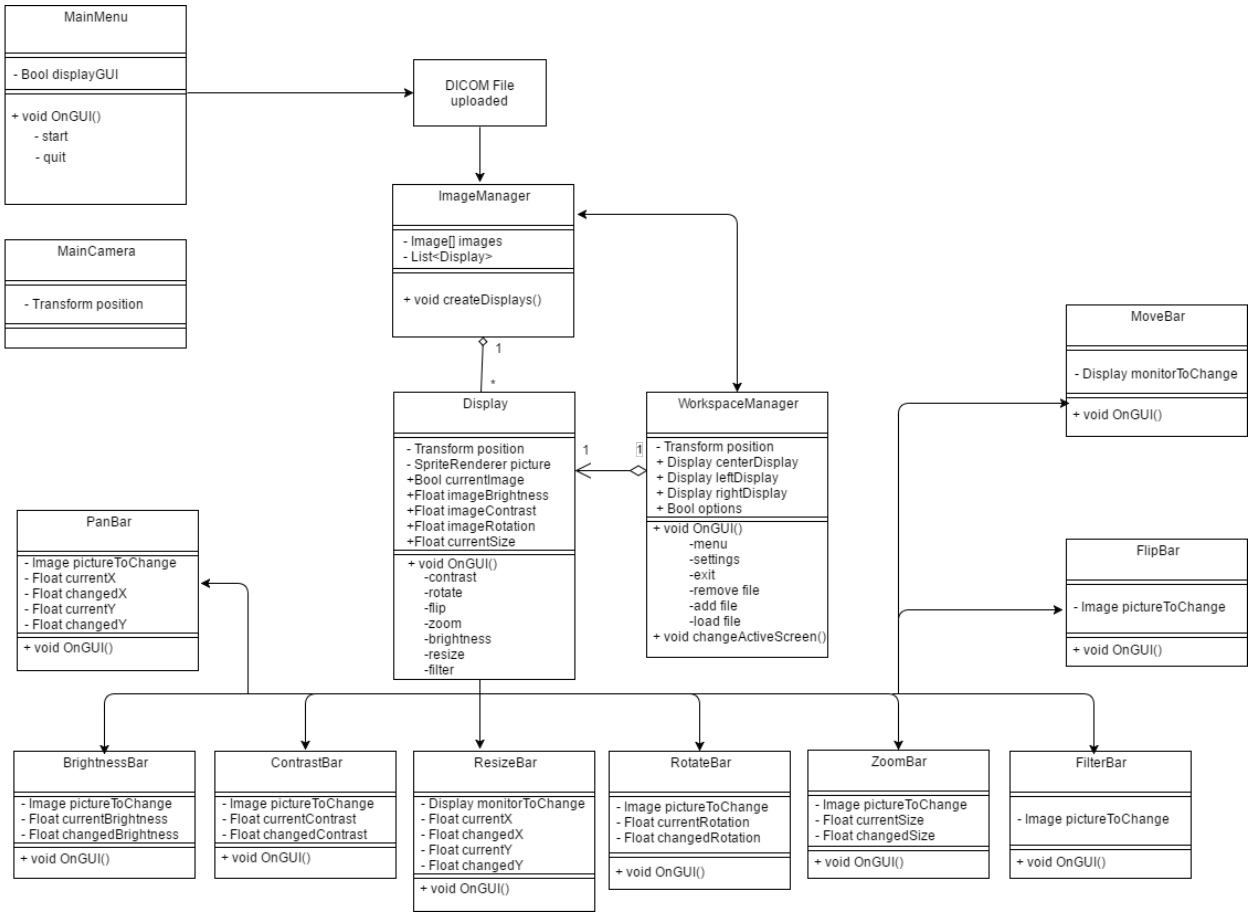

Figure 1. UML Diagram. This diagram represents the relationship, interactions, attributes, and methods involved in each class of our system.

6 CLASS DESCRIPTIONS

6.1 MAINCAMERA

The MainCamera is the default object that is always instantiated and required for all scenes in Unity. The only attribute that will be attached and manipulated with the camera will be its Transform. The direction faced by the camera will be set based on the direction the Oculus Rift headset is facing.

6.2 MAINMENU

This class will represent the first GUI displayed to the user. It will contain two buttons: start and quit. It will also have a Boolean attribute `displayGUI` that will initially be set to true. The `OnGUI()` function will create the two buttons (start and quit). By selecting quit, the user will terminate the program. When selecting start, the class will send a signal to a program responsible for loading and decoding a DICOM file. Once this occurs, the `displayGUI` attribute will be set to false and the MainMenu will no longer be visible.

6.3 IMAGEMANAGER

The ImageManager script will initially take all the images from the DICOM file and assemble them into an array on images. From this array, a Display class will be created for each image in the array and then added to the display list. This will be done by the `createDisplays()` function. To change the currently selected image display (the image directly in front of the user) we will have two functions: `goToLeft()` and `goToRight()`. These functions will initially work with the mouse scroll wheel and change the current image to the one on the right or left depending on the input given by the user. To do so, it will change the attributes of each involved Display's Transform to move everything to the specified location.

6.4 WORKSPACEMANAGER

The WorkspaceManager is a script that will be responsible for which monitors are displayed and additional actions the user can make to the Displays at their disposal. The WorkspaceManager will have a Transform attribute along with three Display attributes. The Display attributes will represent the three current Displays that the user can see and interact with. The Boolean options represent whether or not the other options in the workspace are visible to the user. When selecting the menu button generated by the `OnGUI()` function, the other buttons (settings, exit, remove image, add file, and load file) will become visible to the user. The settings button will allow the user to change personal settings in the workplace (to be implemented and decided on later) and the load button will let the user load additional DICOM file images into the system. The remove file button will bring up an 'x' on each of the visible Displays and allow the user to remove them. The add button will let the user add another visible Display to the workplace (for a maximum of three). The user will be able to select which display they want from the list of displays in the ImageManager. Selecting exit will result in the program terminating. In addition. If the user selects the menu button again, then the other buttons will collapse and the options attribute will be set to false.

6.5 DISPLAY

The Display scripts will be attached to the actual objects that will be displayed to the user (the screens with an image). Each Display class will have a Transform to indicate its position. They will also have a SpriteRenderer attached to them as well, which will hold the image that the Display contains. There will also be a Boolean variable called `currentImage`. The `currentImage` attribute will indicate whether or not that Display is currently being displayed to the user. For each Display, there will be an `OnGUI()` function which will generate a series of buttons: contrast, rotate, flip zoom, brightness, filter, and pan. Selecting each of these buttons will display a corresponding class that will manipulate the desired attribute.

6.6 BRIGHTNESSBAR

The BrightnessBar will be a class that is responsible for manipulating the brightness of an image. The class will be given the image from the Display class to manipulate as one of its attributes. It will also store the current value of the image's brightness as a float in the `currentBrightness` attribute. The `OnGUI()` function will display a slider that can be manipulated by dragging it. The value of the `changedBrightness` attribute will change directly with the position of the slider. The initial value of the `changedBrightness` will be the same as the `currentBrightness`. The user will have the ability to select "apply" or "cancel" which will change the `currentBrightness` to the `changedBrightness` or keep the `currentBrightness` and discard the `changedBrightness`.

6.7 CONTRASTBAR

The ContrastBar will be a class that is responsible for manipulating the contrast of an image. The class will be given the image from the Display class to manipulate as one of its attributes. It will also store the current value of the image's contrast as a float in the `currentContrast` attribute. The `OnGUI()` function will display a slider that can be manipulated by dragging it. The value of the `changedContrast` attribute will change directly with the position of the slider. The initial value of the `changedContrast` will be the same as the `currentContrast`. The user will have the ability to select "apply" or "cancel" which will change the `currentContrast` to the `changedContrast` or keep the `currentContrast` and discard the `changedContrast`.

6.8 RESIZEBAR

The ResizeBar will be a class that is responsible for manipulating the dimensions of the Display (monitor). The class will be given the Display as one of its attributes. Though the image itself will not be manipulated, it is necessary for helping with viewing the image. It will also store the current value of the Display's width and height as a float in the `currentX` and `currentY` attributes. The `OnGUI()` function will display two sliders that can be manipulated by dragging them. The value of the `changedX` and `changedY` attributes will change directly with the position of the sliders. The initial value of the changed attributes will be the same as the current attributes. The user will have the ability to select

"apply" or "cancel" which will change the `currentX` and `currentY` to the values of `changedX` and `changedY`, or keep the current values and discard the changed values.

6.9 RotateBar

The RotateBar will be a class that is responsible for manipulating the rotation of an image. The class will be given the image from the Display class to manipulate as one of its attributes. It will also store the current value of the image's rotation as a float in the `currentRotation` attribute. The `OnGUI()` function will display a slider that can be manipulated by dragging it. The value of the `changedRotation` attribute will change directly with the position of the slider. The initial value of the `changedRotation` will be the same as the `currentRotation`. The user will have the ability to select "apply" or "cancel" which will change the `currentRotation` to the `changedRotation` or keep the `currentRotation` and discard the `changedRotation`.

6.10 ZoomBar

The ZoomBar will be a class that is responsible for manipulating the zoom of an image. The class will be given the image from the Display class to manipulate as one of its attributes. It will also store the current value of the image's zoom as a float in the `currentSize` attribute. The `OnGUI()` function will display a slider that can be manipulated by dragging it. The value of the `changedSize` attribute will change directly with the position of the slider. The initial value of the `changedSize` attribute will be the same as the `currentSize`. The user will have the ability to select "apply" or "cancel" which will change the `currentSize` to the `changedSize` or keep the `currentSize` and discard the `changedSize`.

6.11 FilterBar

The FilterBar will be a class that is responsible for manipulating the filter on an image. The class will be given the image from the Display class to manipulate as its attribute. This class will have a set of filters that can be applied to the image, and the `OnGUI()` function will create a button for each filter we wish to implement.

6.12 PanBar

The PanBar will be a class that is responsible for manipulating the position of the image inside of the Display (monitor). The class will be given the image from the Display class as one of its attributes. It will also store the current value of the Display's `positionX` and `positionY` as floats in the `currentX` and `currentY` attributes. The `OnGUI()` function will display two sliders that can be manipulated by dragging them. The value of the `changedX` and `changedY` attributes will change directly with the position of the sliders. The initial value of the changed attributes will be the same as the current attributes. The user will have the ability to select "apply" or "cancel" which will change the `currentX` and `currentY` to the changed and change (respectively) or keep the current values and discard the changed values.

6.13 FlipBar

The FlipBar will be a class that will be responsible for flipping the image on the horizontal or vertical axis. The class will be given the image from the display as one of its attributes. The `OnGUI()` function will generate two buttons, a “Flip on X” button and a “Flip on Y” button. Selecting either one will flip the image along the desired axis.

6.14 MoveBar

The MoveBar will be a class that is responsible for manipulating the position of the Display (monitor). The class will be given the Display as its attribute. Since we have access to the actual display, we will be able to access and manipulate the individual attributes of its Transform (x, y, z positions). The `OnGUI()` function will display a rotator that will allow the Display to be moved to wherever the user wants. The initial value of the changed attributes will be the same as the current attributes. There will also be a button “OK” which will exit the MoveBar GUI and keep the changes.