CMPT 371 – TEAM 3 DESIGN DOCUMENT

Virtual Reality Medical Imaging Software with Luxsonic Technologies Inc.

FEBRUARY 17, 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 we chose it over other architectures that we considered. In the Unified-Modeling-Language (UML) 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 depict how they will interact 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 with, and affect.

Unified Modeling Language (UML): A general purpose, developmental modeling language that is used as a standard for visualizing the design of a software system.

3 Architecture Description

The architecture that will be 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 being involved or affected. This means dependencies for each class are reduced and each class performs a very specific function. Having this as our design choice will provide 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 being used. Since the Unity environment relies heavily on object creation, everything created in Unity is an object. All objects in Unity have their default components such as Transforms (positions) attached to them. Because these objects are all independent of 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, the team fully expects to create additional functions and potential classes not currently shown in the UML diagram. Since the technology and hardware is new to many members of the team, changes during implementation will occur. Minimizing these changes are ideal, but having an independent components architecture provides 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 specific task, they can easily be reused in situations that require similar functionality. 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 coupling. The reduction in dependencies allows us to work on the project more effectively. The team 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. 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.

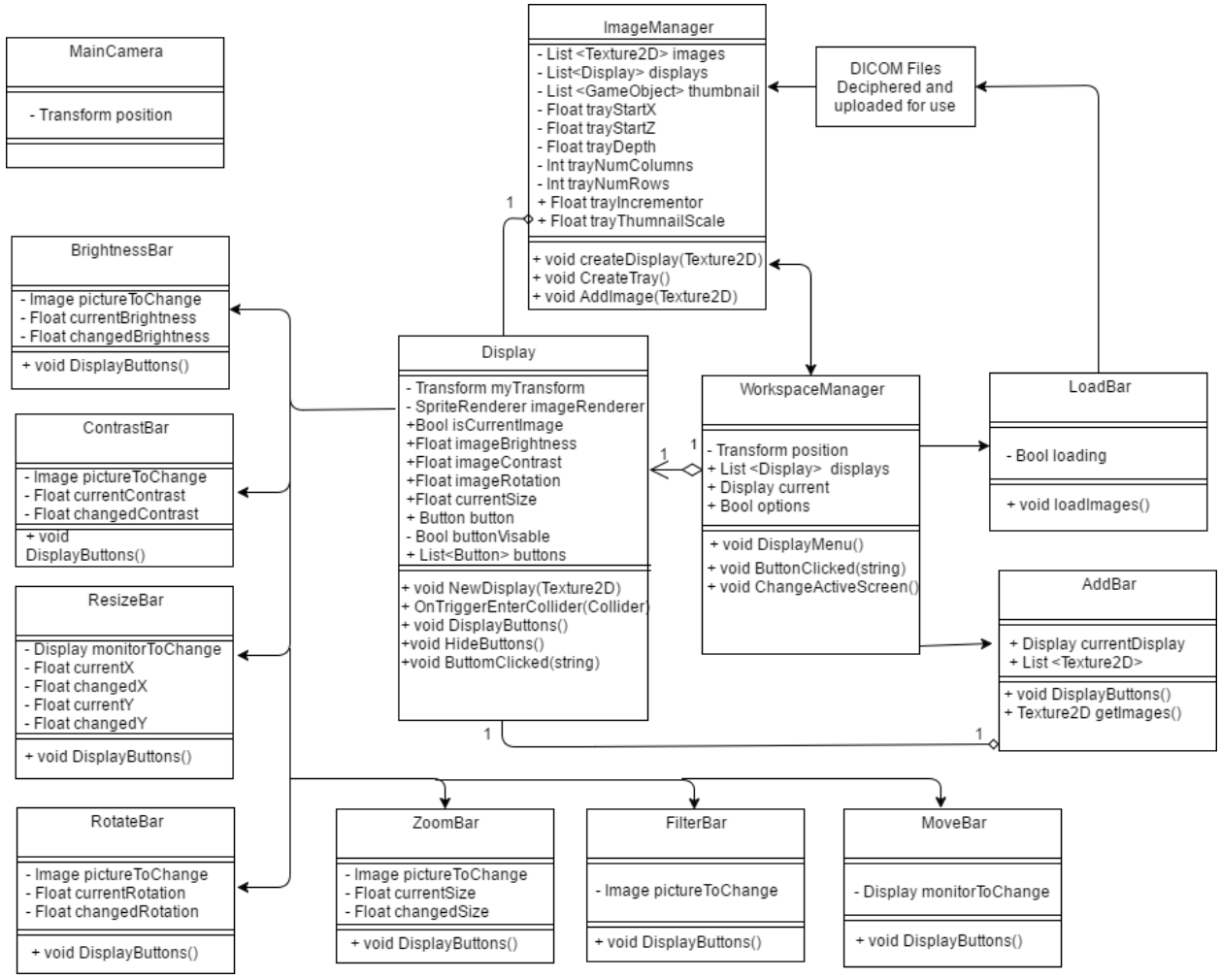

Figure 1. UML Diagram. This diagram represents the relationship, interactions, attributes, and methods involved in each class of our system.

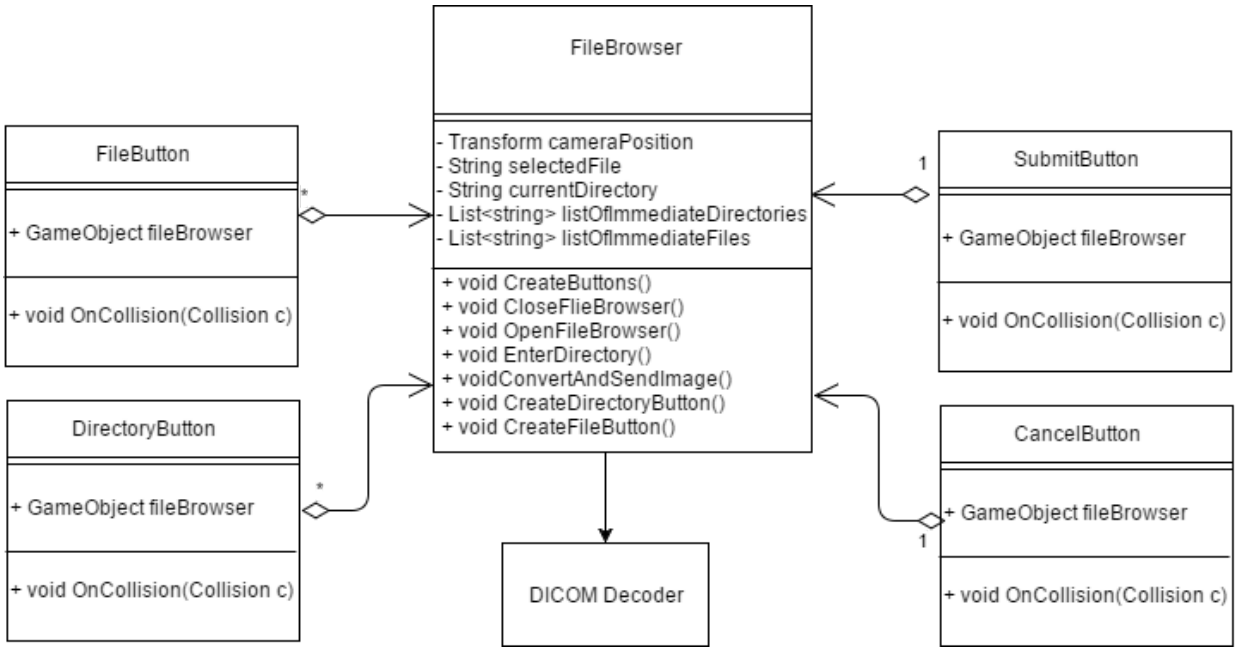

Figure 2. UML Diagram of the FileBrowser. This represents the classes that will be created by and interact with the FileBrowser.

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 of the Oculus Rift.

6.2 ImageManager

The ImageManager script will initially take all the deciphered DICOM images and assemble them into a list of Texture2Ds. From this List, a Display class will be created for each image in the List and then added to the display list. This will be done by the `createDisplays()` function. The ImageManager will share its list of Displays with the WorkspaceManager.

6.3 AddBar

The AddBar class is responsible for adding images to blank displays or changing the image of the current display. Its `currentDisplay` attribute is the display that has been selected by the WorkspaceManager to receive an image. It contains the list of Texture2D images as an attribute from the ImageManager. This list will be retrieved through the `getImages()` function. The

`DisplayButtons()` function will then generate a series of buttons that represent the list of images that the user can select from. Once selected, that image will be added as the image to the `currentDisplay` attribute.

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 component along with the List of Displays from the Image manager. The Display List will represent the Displays that the user can see and interact with. The `currentDisplay` attribute is the Display currently in front of the user that can be altered. 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 `DisplayMenu()` 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 bring up a loading screen and create a new LoadBar class. 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 through the AddBar class. Selecting exit will result in the program terminating. 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 `isCurrentImage`, which will indicate whether or not that Display is currently being displayed in front of the user. For each Display, there will be a `DisplayButtons()` function which will generate a series of buttons: contrast, rotate, zoom, brightness, and filter. 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 `DisplayButtons()` 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 `DisplayButtons()` 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 `DisplayButtons()` 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.

6.9 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 `DisplayButtons` () 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 `DisplayButtons()` 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` 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 `DisplayButtons()` function will create a button

for each filter we wish to implement.

6.12 FileBrowser

The FileBrowser will be a class that the user can call upon to search for DICOM files to load into the system. We want the FileBrowser to be at a fixed position to the user, so it will have to adjust to the Oculus Camera's position. In order to do that, it will have to have a reference to the camera's Transform position with the `cameraPosition` attribute. The file browser will have the string attributes `selectedFile` and `currentDirectory` to store the path of the file selected by the user and to store the path of the current directory. The attributes `directoryList` and `fileList` will store a list of pathnames to the directories that are in the current directory along with the files that are present. This class will include several functions for generating and navigating the simulated directories. The functions `CreateDirectoryButton()` and `CreateFileButton()` will create directory and file buttons for the user to select. Directory buttons will allow the user to enter the directory corresponding to that button. The file buttons will store the path of the selected file in the `selectedFile` attribute. The `CreateButtons()` function will call both the `CreateDirectoryButton()` and `CreateFileButton()` to generate the number of files and directories present in the current directory. The `CloseFileBrowser()` and `OpenFileBrowser()` functions will either allow the FileBrowser to be visible or invisible to the user. The `ConvertAndSendImage()` function will convert a `selectedFile` to a Texture2D and send it to the ImageManager for use. This function will later send the file information to the DICOM Decipher once implemented. The `EnterDirectory()` function will be called to enter and change the `currentDirectory` to the one corresponding to the DirectoryButton selected by the user.

6.13 FileButton

The FileButton class will represent the script attached to the buttons that represent files. It will contain a reference to the FileBrowser through the `fileBrowser` attribute. The only function that will be used is `OnCollision()`, which will detect if the user selects it and then sends the file path for storage in the FileBrowser.

6.14 DirectoryButton

The DirectoryButton class will represent the script attached to the buttons that represent directories. It will contain a reference to the FileBrowser through the `fileBrowser` attribute. The only function that will be used is `OnCollision()`, which will detect if the user selects it and then update the FileBrowser with information on the new directory that will be entered. The `EnterDirectory()` function will be called on the FileBrowser.

6.15 SubmitButton

The SubmitButton class will represent the script attached to the select button. It will contain a reference to the FileBrowser through the `fileBrowser` attribute. The only function that will be used is `OnCollision()`, which will detect if the user selects it and then tells the FileBrowser to call

it's `ConvertAndSendImage()` function.

6.16 CancelButton

The CancelButton class will represent the script attached to the cancel button. It will contain a reference to the FileBrowser through the `fileBrowser` attribute. The only function that will be used is `OnCollision()`, which will detect if the user selects it and then calls the `CloseFileBrowser()` function on the FileBrowser.