CMPT 371 – TEAM 3 DESIGN DOCUMENT

Virtual Reality Medical Imaging Software with Luxsonic Technologies Inc.

MARCH 19, 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.

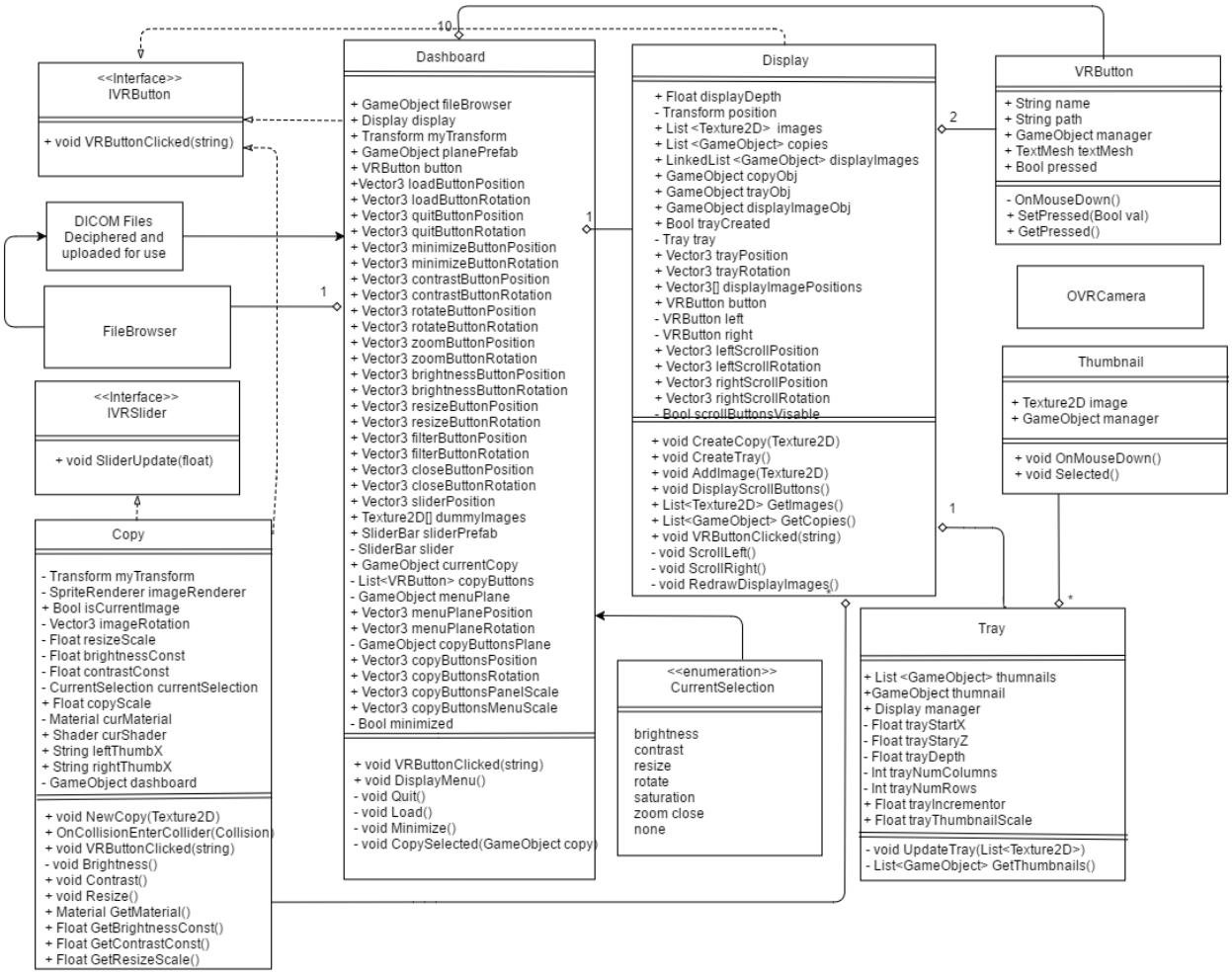

Figure 1. UML Diagram. This diagram represents the relationship, interactions, attributes, and methods involved in each class of our system.

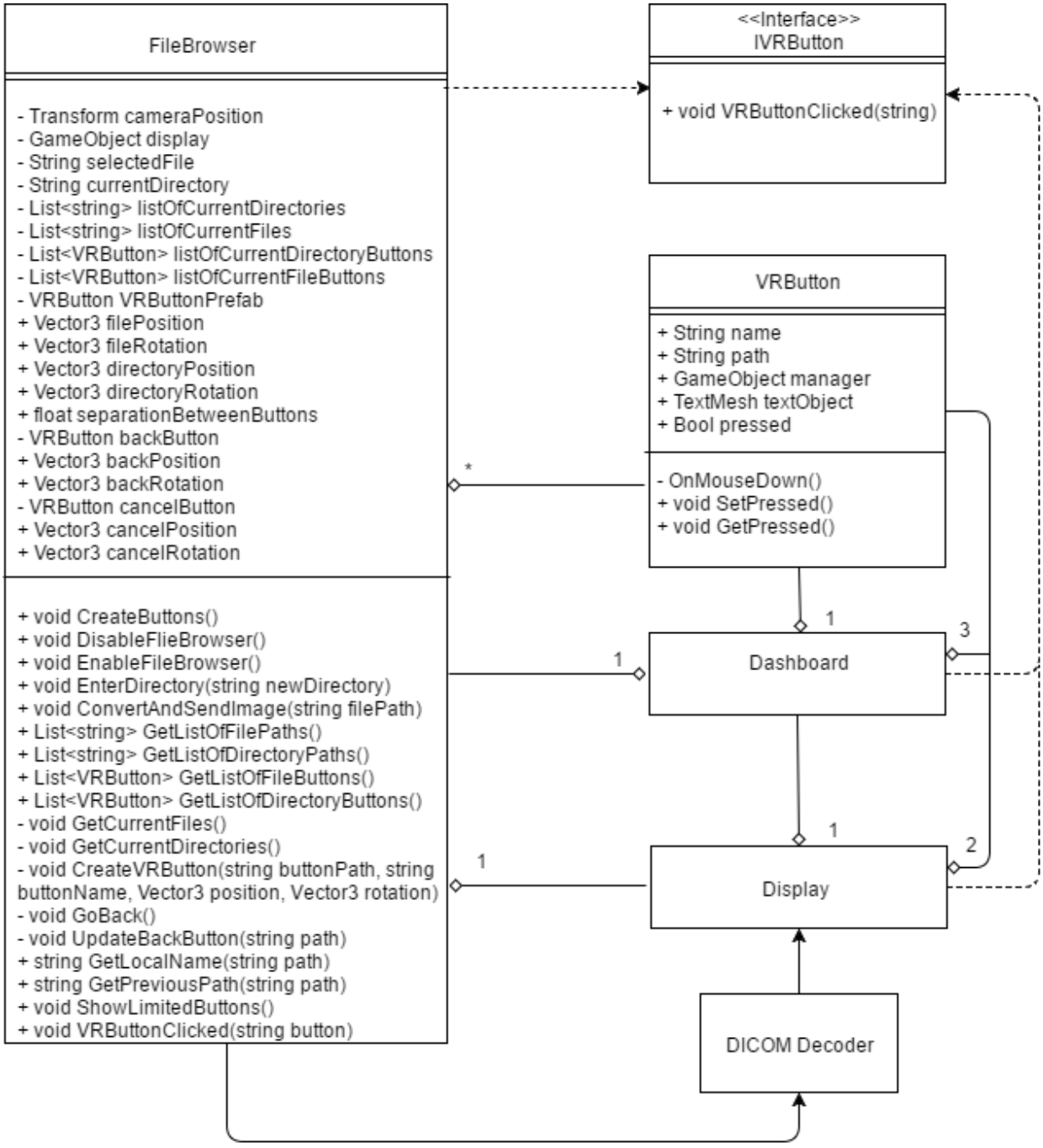

Figure 2. UML Diagram of the FileBrowser. This represents the classes that will be interacted with by the FileBrowser.

6 Class Descriptions

6.1 OVRCamera

The OVRCamera is the camera used by the Oculus Rift to see the virtual world.

6.2 IVRButton

IVRButon is an interface implemented by the Display class. It provides an `IVRButtonClicked()` function that will be called whenever a button created within the Display, Dashboard, or FileBrowser class is selected.

6.3 VRButton

VRButton represents the script associated with an interactable 3D button in the virtual world. In order for a class to create and use this class, it must also implement the IVRButton interface. This class contains two string variables: `name` and `path`, and a ButtonType variable called `type`. `name` represents what the buttons' type as a string, while `type` is an immutable representation of the button's function , which will determine what function will be executed when it is pressed. The `path` will represent an actual path location associated with the button. Only buttons with the name "File" or "Directory" will have a path that is not null. The GameObject attribute `manager` will be a reference to the gameobject that the button is a child of. The TextMesh `textObject` represents the 3D text that is a component of the button. The boolean attribute `pressed` will indicate if the button has been pressed or not. There are only three functions associated with this class. The first is `OnMouseDown()`, which is used to simulate the actions that occur when the button is pressed through the use of the mouse. This is only used for testing purposes and will ultimately be removed in the final build of this project. The second function `SetPressed()` takes in a boolean value and sets the `pressed` attribute to it. If it is true, then a call is made to another function depending on the name of the button. The last function `GetPressed()` will return the value of `pressed`.

6.4 Display

The Display script will initially take all the deciphered DICOM images and assemble them into a list of Texture2Ds called `images`. From this List, the user can look at and browse each image that is present and be able to load more images, quit the program, minimize the Display, or create new Copies from each image present. The Display class will have three lists in total. The first one is the `images` list, which will just store a Texture2D for each image loaded into the system. The second list, called `copies`, will store a reference to each Copy created. The last list, `displayImages`, is a doubly linked list that contains the GameObjects which each image is associated with. The user will be able to scroll through these images, but since the Display could contain many images, the user will only be able to view the ones that are set to active. Additionally, the Display will contain an attribute called `tray`, which will be a reference to the Tray.

There are three GameObject attributes contained within the Display class for storing prefab objects. They are `copyPrefab`, `trayPrefab`, and `displayImagePrefab`. The `copyPrefab` will contain a prefab of the GameObject associated with the Copy class, the `trayPrefab` will be a

prefab of the Tray class, and the displayImagePrefab will contain the prefab used to create the GameObjects in the diaplayImages list.

Two Boolean attributes are also present. The trayCreated attribute will initially be set to false, and is set to true once an image is added and the tray is created. The next Boolean attribute is scrollButtonsVisable which is used to determine if there is enough images in the displayImages list for the user to actually scroll through.

To make sure everything generates in the right positions, there are several Vector3 attributes that are contained within the Display class. They are leftScrollPosition, leftScrollRotation, rightScrollPosition, rightScrollRotation, trayPosition , and trayRotation. These values determine the position and rotation of the Tray class and display scroll bar buttons within the virtual world. To make sure that the images in displayImages are in the correct position, we have a Vector3 array called displayImagePositions.

The last attribute in the Display class is button, which is of type IVRButton from the interface class.

When the user selects an image to manipulate (on the Display or the Tray) a Copy class of the image will be created. This will be done by the CreateCopy() function, which takes in a Texture2D as an argument to produce the Copy from. The Display will also contain a CreateTray() function which will create the Tray from the list of images the Display contains. The AddImage() function will add a new Texture2D to the images list given to it from the DICOMDecoder (once it's implemented). It will create a new GameObject in the displayImages list containing that image. If this is the first image added in the program, then it will create the Tray with the CreateTray() function. There are also two functions called GetImages() and GetCopies() which return a list of Texture2Ds and Copy GameObjects within the system (respectively). The function createScrollButtons() will create both the left and right scroll buttons to browse through the list of GameObjects in the displayImages list. The VRButtonClicked() function is an implementation of the IVRButton interface function which takes in a string as an argument which represents the name of the object that is clicked. The scrollLeft() and scrollRight() functions will move and change the position of the images in the displayImages list as to which images are displayed to the user. The last function, redrawDisplayImages() will set which GameObjects in the displayImages list are visible to the user.

6.5 DASHBOARD

The Dashboard is a script that will be responsible for showing the Display and Tray class. It also contains three buttons that user can use to quit the program, load a new DICOM file, or minimize the Dashboard. The Dashboard will have a several attributes. The first is a Transform called myTransform which will store the position of the Dashboard. The Dashboard will also contain attributes which will reference the Display and the GameObject associated with the FileBrowser called display and loadBar (respectively). There will be an attribute in the Dashboard class called button, which is of type IVRButton from the interface class. From this, there will be three IVRButton attributes called loadButton, quitButton, and minimizeButton. There will be an array of ButtonAttribute instances, holding the parameters for instantiating specific buttons, such as their text and position. A Boolean attribute, minimized, will represent whether or not the Display and Dashboard are visible to the user. Until the FileBrowser class is implemented, there will be a Texture2D list called

dummyImages which stores random images into it for use in the Display.

The DisplayMenu() function will be called on start to create the loadButton, quitButton, and minimizeButton using the InstantiateButton() function, as well as buttons for manipulating Copy objects. The VRButtonClicked() function is an implementation of the IVRButton interface function which takes in a ButtonType enum as an argument which represents the type of button that was triggered. This function will call on a function corresponding to the name of the button sent to it. When selecting the minimize button generated by the DisplayMenu() function, the other buttons (exit, and load) will become invisible to the user through the use of the Minimize() function called through VRButtonClicked(). The Load button will bring up a new FileBrowser class (once implemented) through the Load() function. Currently this function takes a random image from the Resources folder to load into the program. Selecting quit will result in the program terminating by way of the Quit() function.

The dashboard also contains the Copy manipulation buttons such as contrast and brightness, as well as a list of all currently selected Copies. When one of these buttons are clicked, it will send a message to all selected copies representing which manipulation to perform on them. The Copy will then make the manipulation selected on the image attached to it.

6.6 Copy

The Copy scripts will be attached to the actual objects that will be displayed to the user (the screens with an image). Each Copy will represent a 'copy' of a selected image that the user can clone and manipulate. The Copy 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 Copy contains. There will also be a Boolean variable called isCurrentImage, which will indicate whether or not the image has been selected for manipulation.

There are several functions that the copy class will contain. The first is NewCopy() which serves as a constructor for the class which will take in a Texture2D. The copy class will utilize Unity's built in OnCollisionEnter() function, which serve as a way to determine if the user has selected the image for manipulation or deselected it by changing the isCurrentImage attribute. Functions Resize(), Contrast(), and Brightness() will adjust the corresponding attribute of the Copy. GetBrightnessConst(), GetContrastConst() and GetResizeScale() are all getter functions to get those values from the copies.

6.7 Tray

The Tray class will hold a list of images and display a thumbnail of each image on its surface. Selecting a thumbnail on the Tray will create a copy of the image to display to the user (the same as selecting an image in the Display class). The Tray class will have a number of attributes. The first attribute will be a list of GameObject prefabs that will represent each thumbnail stored in the tray called thumbnails. In order to create this list, the Tray will need a reference to the thumbnail prefab GameObject which will be called thumbnail. It also contain a reference will contain three float attributes called trayStartX, trayStartY, and trayDepth which will represent the position of the Tray. It will also contain two integer attributes called trayNumColumns and

trayNumRows which will represent the number of columns and rows of image thumbnails in the Tray. Finally, wit will have a reference to the Display called manager.

There are only two function in the Tray class. The first is the GetThumbnails() function, which returns the thumbnails list. The second is the UpdateTray() function, which takes in a list of Texture2Ds from the Display and makes sure the Tray contains a thumbnail for each image that is in the system.

6.8 Thumbnail

The thumbnail class represents the images that will be present in the tray. It contains only two attributes: Texture2D called image and a GameObject called manager. The image will contain the image we want the thumbnail to display. The manager will be a reference to the Tray class. There are also only two classes: OnMouseDown() which activates when clicked and also Selected() which activates when the thumbnail is touched.

6.9 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. It will also have a reference to the Display class through the display attribute. The file browser will have the string attribute currentDirectory to store the path of the current directory. The attributes listofCurrentDirectories and listOfCurrentFiles will store a list of pathnames to the directories that are in the current directory along with the files that are present. It will also contain two lists of VRButtons called listOfCurrentFileButtons and listOfCurrentDirectoryButtons . These will store the file and directory buttons created. In order to create the buttons needed in the FileBrowser, the class will need a reference to the VRButton prefab which is called VRButtonPrefab. There will also be several Vector3 attributes that will hold positions and rotations of the various buttons created. These include: filePosition, fileRotation, directoryPosition, directoryRotation, backPosition, backRotation, cancelPosition , and cancelRotation. Since the back and cancel buttons are static, there will be an attribute referencing both called backButton and cancelButton. The last attribute included in the FileBrowser will be a float called separationBetweenButtons which represents the space between the directory and file buttons.

This class will include several functions for generating and navigating the simulated directories. The GetListOfFilePaths() and GetListOfDirectoryPaths() will return the listOfCurrentFiles and listOfCurrentDirectories attributes. The functions GetDirectoryButtons() and GetFileButtons() will return the listOfCurrentFileButtons and listOfCurrentDirectoryButtons attributes. Directory buttons will allow the user to enter the directory corresponding to that button. The file buttons will send the path of the selected file to the ConvertAndSendImage() . The ConvertAndSendImage() function will get the file from the path and convert it to a Texture2D and send it to the Display for use. This function will later send the file information to the DICOM

Decipher once implemented. The `CreateButtons()` function will call both the file buttons and directory buttons to generate the number of files and directories present in the current directory. It will do this by calling on the `CreateVRButton()` function. The `CreateVRButton()` function takes in four parameters: two strings representing the name and path associated with each button and two Vector3 arguments for the position and rotation. The `EnableFileBrowser()` and `DisableFileBrowser()` functions will either allow the FileBrowser to be visible or invisible to the user. The `EnterDirectory()` function will be called to enter and change the `currentDirectory` to the one corresponding to the DirectoryButton selected by the user. The functions `GetCurrentDirectories()` and `GetCurrentFiles()` will search the current directory for file and directory paths and set the `listofCurrentDirectories` and `listOfCurrentFiles` to what it finds. The function `GoBack()` will update the current directory the user is in to the previous directory. `UpdateBackButton()` will update the path stored in the back button. `GetLocalName()` will take in a string representing a path and then return the word after the last backslash. `GetPreviousPath()` does something similar, but returns everything except the last word of the path. The final button `VRButtonClicked()` is the abstract function from the interface which is called when a button is pressed.