CALIFORNIA STATE UNIVERSITY
LOS ANGELES

Los Angeles, California

Lunar Exploration Web and Mobile Applications

Lunar Mapping and Modeling Portal Android Application

(LMMPAA)

CS496 Senior Design

Prepared by:

Eddie Arevalo
Alvaro Ortiz
Daniel Soto

Table of Contents

1.0 EXECUTIVE SUMMARY:

The Lunar Mapping and Modeling Portal is a website that allows access to and browsing of lunar data collected by JPL. This information is primarily but not limited to elevation maps of the lunar surface. These maps were collected by different lunar missions such as the lunar reconnaissance orbiter(LRO) and Clementine with a variety of instrument types. Many of these maps are digital elevation model(DEM) maps. Further data consists of a more mathematical or historical nature such as sun lighting across the moon, distances or the historical reason of how a crater came to be named.

The LMMP website already had most of this data featured in an intuitive way. When the first meeting arrived we learned what the patron wanted was 3 different projects that coincided closely to the website. The first was a reimplementation of a part of the website that would generate images depending on the light sources on the moon. This was called the lighting tool(LMMPLT). The second was an android application version of the LMMP website. The name for this application was Moon Tours although we referred to it as the LMMP Android Application(LMMPAA). The third was an application either web or local that would allow the user to see 3D real time images of what someone standing on the moon would see. We called this project the Virtual Photograph Generator(VPG).

As time went on there were many changes to what the requirements for these projects should be. the LMMPLT and the LMMPAA consumed most of the time that would have been used on working with the VPG. The postponing of the VPG eventually led to it being dropped as a requirement as it become more of a priority for the other two projects to be completed rather than have three uncompleted projects. In this way the LMMPLT and the LMMPAA were completed on time. These two projects had working versions although they had to be refined to a point that was acceptable for release.

2.0 INTRODUCTION:

The purpose of this document is to give a good understanding in the use and architecture of the LMMPAA. The LMMPLT will be explained in a separate document. This document will be beneficial to anyone seeking to change or maintain the LMMPAA's code. The information for this can be found in three sections. The first is in secton 3.0 User Guide which will foster understanding in the intended use of the LMMPAA. Secondly for an explanation of the architecture and design in which the LMMPAA was implemented the reader should refer to section 4.0 Architecture. Lastly a reflection upon the lessons learned by presenting which approaches worked and which did not the reader should refer to section 5.0 Conclusions. The conclusions section will point out several flawed ideas that we encountered during the building of this software. It would be especially helpful to anyone working on this project to learn from our mistakes. As Otto Von Bismarck once put it "A smart man learns from his mistakes, but a truly wise man learns from the mistakes of others," or as one of our own CSULA professors put it "It is expensive to learn from your own mistakes, but it is free to learn from the mistakes of others."

3.0 USER GUIDE:

The LMMPAA's primary purpose is to show maps of the lunar surface. As such when first opening the application the user will first be brought to the map screen. Here the user will be able to pan across the lunar surface map. From this point there are 7 options in the menu bar. The options are Search, Bookmarks, Layers, Nomenclature, Markers, Minimap and Settings.

3.1 Search:

The search option allows for a user to type or voice a keyword. This will then show results for that keyword in a list by nomenclature, layer or bookmark. By selecting the nomenclature the user will be shown that location on the map. By selecting layer the user will be directed to the details of that layer. By selecting bookmark the user will taken to the location of that bookmark on the map.

3.2 Bookmark:

The Bookmark option takes the user to see a list of the bookmarks brought by the application as well as any bookmarks the user might have added themselves. By selecting a bookmark from this list a more detailed view will appear. This view contains information such as the history, an image and a button to show the location on the map. The user is also able to add their own bookmarks by selecting the add bookmark button.

3.3 Layers:

The Layers option takes the user to a menu screen which allows the user to select which layers, base maps and special options they want to see on the map. The user can add layers by pressing the green plus symbol at the top of the layers section. This will take them to a list which can be sorted by product type, mission, nomenclature and instrument type. By selecting these layers they are added to the map. The base map can be changed by selecting from the drop down list. Special options can be toggled by checking the box next to their name.

3.4 Marker Tools:

The Marker tool allows a user to draw out shapes on the lunar surface. Data over these shapes can be obtained by selecting them. The user has the ability to place a point as well as draw rectangles, circles, polygons, lines, polylines, and free hand polylines or polygons.

3.5 Minimap:

The Minimap option allows for the toggling of the minimap. This minimap shows the users relative location of the screen over the map. This allows the user to know at what position they are even when they are zoomed in. The minimap can also be used to pan across the lunar maps.

3.6 Settings:

The settings option allows the user the change several preferences that they would like the application to use. This allows the user to set cache size and to change the preferred set of units to use(kilometers or miles).

4.0 ARCHITECTURE:

The LMMPAA is composed of 11 modules. These modules consist of the 8 menu options plus the map view and the utilities. The application is based on an MVC pattern where the data is kept static, the guser interface shows the data and the controllers within those user interfaces allow manipulation of the data. All the data comes from the JPL REST services although the larger portion of this data is kept within the application assets folder in order to greatly increase load time during operations. This is done because the load time to load directly from the rest server is not fast enough to keep up with the demand of any enjoyable use. The packages in the code are organized by MVC conventions. The UI is separate from the model and it is separated from the static data. The application can be explained better by the visualization of modules. The way these modules work together is how this architecture will be explained.

The most important aspect of the Moon Tours application is to be able to browse the moon maps. This is the reason that the first thing that the user will see when opening this application is a screen which allows for movement across the map. This map is the main activity of the project as well as the place where all other activities will either manipulate or somehow point to a location on the map. This activity is powered by the ESRI Maps API. The graphics, layer order, opacity and map callouts are handled by this API.

Before the map view can be shown all the data from the xml and json files from the JPL REST service must be extracted by using the utils portion of code. These classes are inside the utils folder. They are called by the map view on startup via an async task. They handle the reading and placement of the data into the model classes. The async task was used in order to allow the user to browse the map while the nomenclature and layers data is parsed from the XML files in the background.

4.1 Singletons:

The filled model classes are held within Data singletons. These singletons function as the data base from which data can be sent to the user interfaces. These can be split into 2 categories. The first consists of the XML and JSON file information. These include the basemaps, config, layer, nomenclature, tile and tiled group Singletons inside the data folder. The singletons support getting information but not writing as this information is static and must not be changed in order to continue allowing the application to work correctly. The second type allows writing as these classes are meant to keep track of the user selected manipulations of the map view. This data can be changed at any time depending on what the user selects. These include the bookmark, mapdata and markerdata singletons.

4.2 Modules Introduction:

The Search, Bookmarks, Layers, Markers, Minimap, Settings and Help modules make up the 9 other modules. These modules manipulate or show details about the main map view. Each one is connected to its own Singleton class which stores their currently selected data. Each one makes use of Android classes such as lists, fragments, activities and gestures. These function as the view and controller in the MVC pattern.

4.3 Search:

SearchableActivity is accessed by beginning a search. This search is not started through a new intent but instead handled by the Android Manifest XML which sends any query in the search to SearchableActivity. SearchableActivity contains one fragment. SearchableFragment. SearchableFragment shows 3 expandable list views either next to or side by side depending on the screen size layouts. This was done in case later a new version wishes to add more fragments are to be added. The search suggestions are shown by the use of SearchSuggestionsProvider.java This class is configured through the Android Manifest. The search widget it configured in searchable.xml inside res/xml

4.4 Bookmarks:

The Bookmarks section is composed of several files that allow this module to fulfill its functionality. Since this section utilizes the Master-Detail Flow Layout for the tablet version, there had to exist a BookmarkListActivity, BookmarkDetailActivity, BookmarkListFragment and BookmarkDetailFragment. These files of course include their appropriate XML layout files. These files were structured in a way, to allow the tablet and phone versions of the application to have distinct layouts. In addition to the original Fragments, in order to incorporate the ability to swipe between pages of the detail containers, a PagerFragment was necessary, which in turn required it’s own PagerAdapter as well. The index of the currently selected Bookmark is stored in the BookmarkData singleton, to allow the application to determine which pages to supply the ViewPager with.

The overall content for the Bookmarks varies since there are two possible categories for what the Bookmarks could be. A Bookmark could be predefined, or it could be user-created.

4.4.1 LMMP Bookmarks

Initially, the Bookmarks section is filled in with predefined LMMP Bookmarks of famous lunar missions and sites. The content for these was gathered from the LMMP Web Service, which provided us with a Json file containing various information about these LMMP Bookmarks. We then converted this Json file into actual Java Objects using model classes. Once we had this information in model classes, we were able to map the information into the BookmarkData singleton.

By using the BookmarkData, we generated the views in the application showing the content of these predefined Bookmarks.

4.4.2 User Bookmarks

An addition to the Bookmarks section was the ability to add Bookmarks. Since we wanted the Bookmarks implementation to allow the user to create their own bookmarks, a custom BookmarkDialogFragment was necessary in order to receive the input from the user and therefore use it in conjunction with the rest of the Fragments, to dynamically display this information. By supplying this custom dialog with a title, subtitle, and description, the user will ultimately create a bookmark. By tapping on “Save”, the user’s newly created Bookmark would be saved into internal storage. Along with the supplied information, the bookmark will save the current map extent, location, and any layers currently selected.

Since we have a separate list for user created bookmarks, we needed to implement the ListFragment and the ListAdapter built specifically for the Expandable List view. This view allows us to display multiple group lists, which could contain any number of child items. This allowed the switching between the predefined bookmarks and the user-created bookmarks, as well as the swiping between both of these lists.

4.4.3 Displaying Bookmarks

In order to share information among the Fragments in Bookmarks, we emphasized the usage of Callbacks. Callbacks are essential since Fragments can't communicated directly with other Fragments, and in order for them to communicate with each other, they have to first communicate with the current Activity. By using Callbacks we were able to implement the "Show on Map" feature, which would use the value for the selected Bookmark and send this value back to the Parent Activity, which would terminate and return to the initial MapActivity with a value as a result. The MapActivity would then have a function that would get called whenever a certain value was returned from any other section of the application. In this function, it checks to see if the user decided to show a bookmark.. If the user returned a value, then this function would call another function called "ShowBookmark", that would display the bookmark associated with that index. This was done by simply calling the BookmarkData singleton with this index and extracting that bookmark's layers, basemap, resolution, possible callout. Esri's built in functions would then allow us to include these features into the map view by calling several functions. We also had to make calls to the MapData singleton, letting that data holder know about the new changes to the map’s content.

4.5 Layers:

The Layers Activity is started within the Map Activity. LayersActivity.java checks if the device is a widescreen or small screen. In both cases LayersFragment.java is added to the layout view by a fragment manager. The reason for using the fragment manager is to be able to replace the fragment later if necessary specifically during drag and drop operations. If the screen is large than the The LayersSelectionFragment.java is also added to the layout. Layer Fragment contains the code for changing the basemap, setting and unsetting the special layers. These changes are detected by the callback methods that are implemented inside the LayersActivity.java class when the screen is large. If the screen was detected to be smaller the Layers Selection fragment is instead opened inside a new activity called LayersSelectionActivity. This activity provides its own callback methods similar to those for the layers selection inside LayersActivity.

4.6 Nomenclature:

NomenclatureBrowseActivity is started from MapActivity through a selection in the menu. This class provides the layout in which the sort by spinner is shown. Depending on what the user selects NomenclatureListFragment will be loaded with the same list adapter but with a differently sorted map consisting of the nomenclature data. The default selection is alphabetical but when the selection is changed then the fragment manager replaces this fragment. For this reason the first time this activity is created the fragment manager must add this fragment to the layout. On a large screen NomenclatureDetailFragment is added to the layout. If a small screen was detected then an onclicklistenerinside NomenclatureDetailFragment will open a NomenclatureDetailActivity which in turn opens the NomenclatureDetailFragment using the FID of whichever feature was selected.

4.7 Marker Tools:

The files related to the marker tools are: Marker.java, MapActivity, MarkerData, MoonMapViewListener, and MarkerDialogFragment. Marker functions mostly as a model class while the rest of the mentioned classes use functions to help create the different markers. The flow for the marker tools starts at the MapActivity then to the MarkerData where it stops. After MarkerData, the MoonMapListener listens for input and, depending on input, sends information to MapActivity that leads to the MarkerDialogFragment or to an instance of MarkerData.

4.7.1 Marker.java

The Marker class has a list of types that it can be. The list is essentially the list of geometries that the application gives the users to draw. The different fields in the class are: type, used to describe the instance of the marker; geometry, which is the actual Geometry used to draw out the marker; and the graphicId used to find the marker if ever needed. Marker only has one constructor which takes in a Type and a Geometry.

4.7.2 MapActivity

In MapActivity we have many functions that are not relevant to the marker tools, but the relevant ones are: markerModeOn, markerModeOff, removeMarker, and showMarkerDialog.

4.7.2.1 MarkerModeOn function

This function calls is a helper method that calls MarkerData's markerModeOn function. This function first calls for an instance of the MarkerData and calls MarkerData.markerModeOn passing the markerType that was sent to the markerModeOn function.

4.7.2.2 MarkerModeOff function

This function is a helper function method that also makes changes to the menu items. This function first calls for an instance of the MarkerData and calls its markerModeOff function. It then updates the marker if it is of type Polygon. It also set the markerMenuItems to visible while making the markerOffMenuItem, which is the button that turns off the selected marker tool invisible.

4.7.2.3 RemoveMarker function

This method takes in a graphicId and removes the graphic from the MapActivity. It first gets the graphics layer from MapActivity and removes the graphic, then removes the marker from the instance of MarkerData.

4.7.2.4 ShowMarkerDialog function

Takes in the graphicId for the marker whose data will be shown in the MarkerDialog, and displays the MarkerDialog.

4.7.3 MarkerData

4.7.3.1 CreateMarker function

This function takes in a point and creates a marker while also create the Geometry for the marker based on the markerType.

4.7.3.2 UpdateMarker function

This function updates the graphic for the given marker using the given point. This function is called from the onDragPointerMove function in the MoonMapListener class.

4.7.3.3 MarkerModeOff function

Adds the given marker to the list of markers and clears the points list and resets the markerType.

4.7.3.4 AddPoint function

Adds a point to the marker by adding the point to the list, points.

4.7.4 MoonMapViewListener

Handles onSingleTap, onDragPointerMove, onDragPointerUp, and onFling.

4.7.4.1 onSingleTap

First checks to see if a marker tool is active. If not, then it attempts to see if the user tapped on an existing marker in order to show a dialog for the marker. Otherwise, it attempts to see if the active marker uses onSingleTap to create the marker.

4.7.4.2 onDragPointerMove

This function is overriden and is only called when a marker that uses the drag gesture is active. This function updates the marker, changing different properties of the geometry.

4.7.4.3 onDragPointerUp

This functions is overriden and is called when a marker that uses this function is active. This function ends automatically closes the marker calling markerModeOff.

4.7.4.4 onFling

This function is overriden because it is often called when the user attempts to do a drag gesture. When the marker tools are on, this function ensure the screen does not move while the user is attempting to make a drag gesture.

4.7.5 MarkerDialogFragment

This class displays the dialog for the different markers. There are several if statements that handle every single marker type to display data specific to the marker type.

4.8 Minimap:

The Minimap interface is composed of two features. The first feature is the actual minimap which is essentially a view with a few extras modifications to transform it into an actual minimap. This was created as a custom View class in order to incorporate additional functions. The second feature in this interface is the crosshair view. This again, is a different custom View class that allows it to have the functionality of a crosshair, along with the constant updating of the latitude and longitude values.

4.8.1 Minimap View

The first feature is the actual minimap which is displayed in the corner of the map view. The minimap was implemented in a way that allows it to use the map view coordinates, and translate these coordinates onto minimap view coordinates. Functions belonging to the minimap allow such translations to occur during sudden changes in the map view extent. Therefore, the minimap detects whenever the user moves around the map view, and redraws the minimap view to fit the new translated values. A rectangular box is created with Android’s graphics library, to show the extent of what the user is currently viewing on the overall map. The more zoomed in the user is, the more useful the minimap tool is. The user can move to a different area of the map by tapping on the area he/she wants to move to, in the minimap.

4.8.2 Crosshair & Coordinate Values

The second component of the minimap interface, is composed of the crosshair and the coordinate values at this crosshair. The crosshair also contains the latitude and longitude values of the center of the map view. These values pertain to the current location on the map view and change whenever the user moves around the view, whether it be through the minimap or the map view.

4.8.3 Minimap Interface Toggle ON/OFF

An option is available in the Action Bar, which allows the user to toggle the overall minimap interface ON or OFF. This makes both the minimap and the crosshair features both visible or invisible. Since the Minimap class holds both the minimap view and the crosshair view, it facilitates the action to set the visibility of these two features.

4.9 Settings:

The settings is made of 4 files: SettingsActivity, SettingFragment, arrays, and preferences. SettingsActivity and SettingsFragment fall under the controller category, the arrays file falls under the model category, and the preferences file falls under the view category.

4.9.1 SettingsActivity

This is the activity that is called from the action bar in the main Activity. All this Activity does is call the SettingsFragment.

4.9.2 SettingsFragment

This Fragment is called by the SettingsActivity. The EditTextPreference that appears in the fragment is set to the name of cacheSizePreference. The cacheSizePreference has its OnPreferenceChangeListener set to an anonymous inner class.

The inner class for OnPreferenceChangeListener attempts to parse the value, named newValue, that the user inputs into the EditTextPreference.The inner class contains a try and catch blocks that catches NumberFormatExceptions that may occur when attempting to parse the newValue Object into a double value. Inside the catch block, if newValue is an empty string then a Toast is displayed with the string from R.string.empty_field_exception otherwise the string displayed on the Toast is from R.string.number_format_exception. Inside the try block, there is an if statement that displays a Toast, with the string from R.string.value_too_low, if newValue is below the constant double CACHE_SIZE_MUST_BE_OVER. Outside the if statement, the new cache size is set and cacheSizePreference's summary is set to the string R.string.pref_cache_size_summary and is given the new cacheSize to display.

After cacheSizePreference's Listener is set, the ListPreference in the Activity, named unitsPreference, has its OnPreferenceChangeListener set to an anonymous class.

The OnChangeListener changes cacheSizePreference's summary to the current status using the string R.string.pref_units_summary.

4.9.3 Arrays

The arrays.xml file contains the array for the Units preference. If there is any need to change the array, it can be done through this file.

4.10 Help

The help section relies on the HelpActivity, and the help folder under assets.

4.10.1 HelpActivity

The MapActivity calls the HelpActivity under the onOptionSelected function. In the HelpActivity the contentView is set and the WebView is created named helpWebView. The helpWebView's onReceivedError function is overriden to handle an error that is thrown when an html link goes to a specific section of the webpage. After the onReceivedError function is overriden, the webview loads the index.html url.

4.10.2 Help Folder

The help folder contains all the different html pages that can be traversed through the help section. All the html pages are styled using the style.css file which should be the main method of styling the different aspects of the html page. In the help folder there is also the image folder which contains all the images that are displayed in the different pages. The images that are used should be relatively small so that they will look good in a smaller screened phone.

4.11 About

Uses the AboutDialogFragment class. This displays the dialog_about layout to display the version and app versions. There is a function to get the data version from config.json under assets that parses the data and there is also a function that gets the app version from the manifest.

5.0 CONCLUSIONS:

5.1: Reusing Software:

Many lessons were learned from this project but the most important one is about refactoring. Originally an application had been given to us to work with. We attempted to use this code as a base to work off of but this turned into a mess as the previous and new bugs mixed together become almost impossible to fix. The existing bugs were the most difficult to find as we had no idea where they were coming from. Having spent the entire first quarter bogged down we realized that we would have to rewrite the entire application. After the application core was rewritten we added all the modules one by one while testing each one individually. If we had done that from the beginning we would have finished the project much faster. This is one important lesson that we have learned. If the code is already buggy do not add to it. It is time to refactor or remake it.

5.2 Proper Use Of Time:

A second lesson that was learned is how much can be accomplished within a limited set of time while attempting to handle the other courses. As this was not a normal class a specific time to work was not set up and instead work was done as time was available. Usually in the fashion of doing the more immediate course assignments before working on the senior design project. In order to combat this lack of sense in order. We set up goals to accomplish by the week in order to know if we were on track or not. The original quarterly plan quickly fell apart and we were forced to plan by the week although most of the major goals could be moved to fit a better schedule. Having deadlines on this smaller scale and having them due to the professor during meetings pushed us to work harder and have something working in order to show it during that meeting.

5.3 Android Programming:

The third lesson was programming in Android. There was a learning curve in that none of the team had ever worked on Android applications before. This meant that every move by anyone on the team would have to be thoroughly researched first by use of documentation, examples and tutorials. It was quite often that a plan for implementation failed due to a previously unknown part of the plan being unable to be done the way it was planned. Such as not being able to use lists to show certain information or not being able to manipulate views because they were referenced and there was no way to track their movements. This weighed heavily in the amount of time consumed as much work was done but by manner of it being research there would be nothing to show for it. We read the books and did tutorials before beginning the project but this was almost useless as most of what was learned was not encountered in this project. We did however notice many Android programming elements that we would commonly run into and wish we had known ahead of time to focus our study on these elements.

The elements of Android programming that we commonly ran into were plenty. A good list for anyone preparing to work on this project is as follows(in no particular order):

- StartActivityForResult
- Custom ArrayAdapter
- Custom ListAdapter
- Custom ExpandableList Adapter
- Custom DialogFragments

- Singletons
- Parsing Json
- Parsing XML
- ESRI Maps
- Pager
- Fragments
- Callbacks
- Handling different screen sizes
- Search widget
- Touch listeners
- Spinners
- XML String values
- XML styling
- XML widgets
- XML layouts
- LRU cache
- Handling screen rotations
- Handling menu option selections

The most effective way to use these is to make a working simple version in a different project using similar dummy classes. Write down the steps you took to implement it and then follow those steps to redo it inside the main project with the actual classes. This way confusion is minimized and testing runs are shorter as running the dummy project is much faster than running the main program every few minutes.

5.4 Android Testing

Bugs are very common to find during testing but there are specific places anyone working on this project should check. While working on android and its various components the place where bugs are most commonly found is on rotation. Always check on rotation while testing code. Another place where bugs are commonly found is while handling different screen sizes. Always check both the phone and the tablet. While the phone works by using several intents and activities, the tablet will generally work on callbacks. Both are handled separately and while one works the other may not. A third place where bugs are commonly found is on the mapview. If one graphic is showing on the map and a function that applies another graphic is run it will usually cause some sort of bug. A good way to test is to run the application on the phone and tablet at the same time then rotate them.

5.5 Possible Future Changes

There are several features and types of implementations that we either didn’t have the time to implement or did not know about before our implementation began that could improve the application.

5.5.1 Use Of A Database

Use of a database rather than reliance on json and singletons may allow data to persist within the application. This would speed up the application on startup as the json and the xml files would not have to be parsed to refill the singletons with data after the first start up on a device. Currently the data exists in the static singletons which are destroyed by the device when the application is not in the forefront and the device is out of memory. This causes the application to refill the data singletons by parsing the json files on the next startup. By use of a database such as SQLite the application would never have to reparse data except in the case of an application update. The reason this was not implemented in the first place was because we did not know that this existed on the Android until after the singletons were implemented. The second reason was because this version of the application was based on the prototype. This prototype made no use of a database either. The Database would also increase the speed of searches through use of its query system.

5.5.2 Marker Tool Functions

The marker tools allow the selection of locations on the lunar surface and further information gathering over those locations. Currently the marker tools allow the selection of locations and the ability to get the coordinates of those locations. The next step would be to feed these coordinates to the REST services in order to get access to other tools such as the Elevation tool and the Hazard analysis tool. The marker tools a gateway to a host of other functionalities that bring the application much closer to that of the lmmp web application.

5.5.3 Basemap Support

Currently the application supports four of the eight base maps that can be gathered from the REST services. There are four other base maps which do not have the same rectangular extents as the other four. These are the pole maps. These pole maps should be able to switch the extent of the map to a different extent. We were planning on adding these maps but ran out of time. By adding these base maps we would introduce even more interesting locations.