CS1530 – FINAL DELIVERABLE

PROJECT: Cemetery Plotter

MEMBERS:
Jiajie Dang (djjkobe) – Software Engineer
Michael Faller (mdfaller) – Software Engineer
Daniel Kindler (dkindler) – Project Manager
William Rossi (wjrossi) – Quality Assurance
Xiaokai Wang (daytimebat) – Software Engineer

DUE: 24 NOV 2015

2. DESCRIPTION

After months of development, we've really stayed the same course in terms of communication. We continued to use Slack as our primary communication tool. We didn't only use if for communication, but also used it as a notification tool. Any time any update occurred in the project, for example a pull request or a new commit, we would all simultaneously be notified. This helped our whole team continuously stay updated throughout the development process. We always intended on using a tool to better delegate tasks throughout the development period of this project. However, that never happened. Since we were such a small team, we easily handled delegation through our main Slack channel. There was never a true need for us to use a tool like Trello.

Again we faced difficulties doing code reviews and pull requests. Our limited availability for this project made it hard to fulfill timely reviews. If we were all in an office, ready to go at nearly any time, it seems like it would be a great system. It would help with integration, possibly prevent a few bugs, and help us each familiarize with other parts of the codebase. It was impractical to wait most of the time. The benefits of working in branches is also clear, but we still worked on main frequently, which was only feasible with our sparse schedule.

The code itself has become a behemoth compared to many of the other projects we've worked on for school, if not all. It's easy to see how bugs are introduced, that are not discovered until much later. Even with that in mind, it was almost impossible to prevent errors from cropping up; in our case, usually NullPointerExceptions. Any feature we worked on had it's own set of new and unusual behaviors. They usually weren't even that hard to track down. Fixing them in a responsible, thorough, and clean way however, took some thought. The cloud storage was a nightmare because we used a partially completed, mostly undocumented library called parse4j. The difficulties it posed were not clear when the decision to use it was made earlier on. For one thing, it is threaded and our program is not, except for loading the GUI per Swing specifications. We got around this by adding some busy waiting which is terrible, but doesn't really seem to have a major negative effect since the user would expect to wait a moment when loading a file from the cloud.

We tried to maintain the layered approach. It was difficult to keep things separate, especially since we were all new to programming with Swing. If you were to look at our code from a Model-View-Controller perspective, it's likely that our program did not achieve that level of separation. The View and Controller parts are mostly one-in-the-same.

Creating the map was probably the most interesting thing we did. It was a relatively last minute idea, narrowly escaping the cutting-block. It went through a lot of changes in a very short time. We knew we had this detailed map already, the issue was how to

display this giant image in the GUI and be able to select coordinates and draw on it. We tried using a JEditorPane and HTML at first, and it might have worked, except it turns out it doesn't support any useful CSS that we required. We looked at a bunch of other GUI objects, but it turned out the answer came from one of the outdated Oracle Swing tutorials. They had a demo where they extended a JLabel into a ScrollablePicture class that we were able to hijack for displaying the map. Then we were able to override the paintComponent method to include the optional drawing of a location indicator (a green square). It took a bunch of research to figure out what methods to use center the map on the location and how to find the coordinates of a mouse click, but those parts came together nicely. It could have been a lot worse.

As stated earlier, our customer had great expectations for the product. They wished for many features that, inevitably, we would not have time to complete. Thus, one tradeoff we made to ensure we deliver on time was to sometimes tell our customer, "no." Many times in software development the customer may ask for things related to the project that are completely out of scope. In our case, one prime example was that the customer wanted an integrated a map system in which they can enter any given plot and be shown its location. Because of our limited time, I told Robert and Lonnie that this feature would not be able to be accomplished on time. However, we accommodated their needs by including an offline solution. We utilized a high-resolution PNG image of a pre-existing map of the cemetery to act as a reference for plot and section locations. We felt this was a satisfactory compromised and fulfilled the spirit of their requested feature.

We also cut a few other features from the GUI, like being able to filter the people list by user-selectable fields. With that feature out, we also had to remove the ability to search for people by those fields (like phone number). This would have been hard to implement with our current code. To do it, we would have had to extend JList into another class that held our People objects (and made them proper subclasses). It would also be inefficient because we relied on some technical debt. We never changed our ArrayLists to HashMaps. It is doable, but tedious, and we probably would have had to sacrifice other features to sink enough time to properly debug the changes.

Testing for this sprint was completed manually. It was challenging to separate logic from the various listeners and handlers involved with our GUI fixes and enhancements this sprint. It is difficult to program user interaction with a GUI for the sake of testing, and from experience we know that manual testing is an important part of the QA process. As such, user-action-related methods were verified manually rather than by creating unit tests for them. The same process was performed for our various save / update / load methods, our logic being that if a file was not saving / updating / loading correctly, we would easily be able to tell without the need of a unit test. Since this is our final deliverable and we will not be making any more changes to the code, we considered this manual testing an acceptable solution once we were no longer able to break the software through thorough interaction.

The main thing we would do differently is use a lightweight database instead of creating our own file. None of us knew how to use a database (at the beginning of the semester at least) and we had no way to realize how many checkboxes one would tick for this application. It is painfully obvious now.

3. USER STORIES: COMPLETED THIS SPRINT

GitHub Repository: https://github.com/wjrossi/Cemetery-Plotter

1. As a user, I want to be able to save the data to and get the data from the cloud, so that I can backup the data and use the program remotely.
2. As a user, I want to be able to see a map of the cemetery, so that I can have a visual reference.
3. As a user, I want the file to be compressed, so that it doesn't take up as much space in the cloud, or take too long to transfer.
4. As a user, I want to be able to load and store the location of plots, so that I can see them on the map.
5. As a user, I want the location of the plot to be highlighted on the map, so that I can easily see it.
6. As a user, I want to be able to search through the plots, so that I can find a plot quickly.
7. As a user, I want to be able to search through the people in a section and if they have a plot, I want to display their info.
8. As a user, I want to be able to store notes and comments about each plot, so that I can remind myself about any other info.
9. As a user, I want to be able to flag a plot as belonging to a veteran and/or a person who was cremated, so that I can properly catalog it.
10. As a user, I want error messages to pop up if I did something wrong or if I am about to do something risky, so that I don't mess everything up (too easily).
11. As a user, I want to be able to easily start a new cemetery file, so that I can make a new file without risking altering an existing one.

4. USER STORIES: DETAILS

User stories one and two were two critical stories we knew about from the beginning. Cloud storage built on a lot of prior work. It was the last thing we got working and it is not an ideal solution or implementation, but a passable one nevertheless. Compressing the file was also a helpful addition that went in prior to cloud storage. The map is also critical. It's the main feature that anyone would notice. The original idea was some sort of GIS or Google Maps type solution in an embedded browser pane, but that would have required a tremendous amount of research and work. We knew the synagogue already had a very detailed map of the cemetery and decided to utilizing that for our solution. It achieves most of the desired functionality: plot locations can be selected graphically, stored, saved, loaded, and displayed on the map. User stories six and seven were both basically the same. We had lists and we wanted to make sure you could search through them. Eight and nine were important for functionality; data layer items that we should have had in place from the beginning. The last story, adding dialog boxes and warnings, was necessary to protect the user from making mistakes and also to prevent a lot of bizarre errors like attempting to delete a plot when none is selected, for example. The last user story, adding a File->New button was a quick fix for starting new files instead of loading an existing one, clearing it, and saving it as a new one. This way it behaves like any other program.

5. CUSTOMER INPUT

We initially split this project into three main requests from Lonnie and Robert: User Interfaced, Cloud Based, and Informative. After several months of development, we achieved all three major milestones successfully, to the customer's delight. In our initial meeting with Robert and Lonnie, the two executive members of the Congregation, it was decided that the three major improvements needed are:

a. Information was what was stored in the database. This can be the names of the interred, who owns a plot, how much an individual might owe the Congregation, how many vacant plots there are, etc. After months of development we can happily say that we have included all of this information successfully into the project. Hopefully it will be enough for Robert and Lonnie to handle the intense information that running a cemetery incurs.
b. Having everything stored in the cloud was another huge issue. Lonnie, the executive director of the cemetery, was planning to retire and move down to Florida. He wanted to be able to access all the data from Florida without having it be inaccessible to those in Pittsburgh. Our solution was to implement an “Upload” and “Download” to/from cloud options that would let updates be made, and then have them saved or retrieved from the cloud. We integrated Parse to save the files safely. We believe that this method is an effective solution to Robert and Lonnie’s original problem, and believe it will better suit their needs, and help them run a better, more efficient cemetery
c. Finally, GUI. A lot of work was put into this feature, because it seemed to be the most important feature listed by Robert and Lonnie. They both stressed the age of their current DOS program, and how the new application should be one of the twenty-first century. We implemented an immersive user interface that will prove to be more intuitive and easy to use by Lonnie and his team. We made sure that we accommodated for the Congregation's needs by listening to their concerns. One concern was that they wanted a map included in the program to make it easier to spot out and list any plots in question. We hope that this will be sufficient for Robert and Lonnie in the long term.

And finally, another request from Robert was to give him the source code after development. Since Robert has a background in computer science, he was hoping to be able to add to the project and continue to grow it. We as a team decided this request is perfectly reasonable, and will be giving Robert access to the repository for him to contribute to.

6. DEFECTS

Weird behavior while searching in the people list

1. **Reproduction Steps:** Select all cemetery sections, start typing a name of an existing interred person in the people search field.
2. **Expected Behavior:** Other names would disappear as they stopped matching what you are typing and once there is only one option, it is selected, and it's matching plot is selected in the plot list, and the info is displayed in the center.
3. **Observed Behavior:** The non-matches disappeared but a different plotID than the one belonging to the person you searched for is selected and displayed.
4. **Solution:** The JList with the people's name had a ListModel that held the data. While searching a new ListModel was created with the matches, but the JList event listener the responded to selections was looking at the unchanged ListModel containing the full list of plots and was selecting the data belonging to an unlisted interred person. Setting the oldListModel equal to the newListModel after filtering the search results each time made it work.

People list not clearing

1. **Reproduction Steps:** Select some section(s) and search for a person. Then deselect the section(s) or select a different one.
2. **Expected Behavior:** The search field and people list would be reset.
3. **Observed Behavior:** The search results remained.
4. **Solution:** Add a method call to clear the people list in the event listener for the sections JList object, whenever a change in the selection happens.

Adding duplicate plots with New Plot button.

1. **Reproduction Steps:** Select multiple sections and click the New Plot button. Delete any one of the plots.
2. **Expected Behavior:** Unknown. Maybe they would all be deleted
3. **Observed Behavior:** A plot with identical plotID is added to every section selected. Only the single plot selected is deleted. Trying to delete any others results in NullPointerException.
4. **Solution:** Added a check when the New Plot button is pressed that only one section can be selected. If more than one is selected, throw up a dialog to alert the user. This keeps duplicates from being added and avoids the unknown behavior.

Delete plot when none is selected

1. **Reproduction Steps:** Select section(s) and click delete plot.
2. **Expected Behavior:** Doesn't do anything.
3. **Observed Behavior:** A NullPointerException is thrown.
4. **Solution:** Added a check when the Delete Plot button is pressed that a plot must be selected. If none is selected, throw up a dialog to alert the user.