Parker Rowland
Lead programmer
Mice Visualization team
CPSC 430, section 1, Spring 2017
4/18/17

Notes to the next Mice Visualization Program development team

The purpose of this document is to provide additional insight to developers working to further improve the program created by the CPSC 430 section 1 Mice Visualization team.

Currently unimplemented features:

In its current state, the program provides functionality matching most of the original requirements specified in the requirements and test plan documentation, although there are several features, both client-specified requirements and implementation team functional goals, that were not implemented due to time constraints. Features not implemented include:

- A loading indicator used to provide feedback to the user during file load / data processing operations
- The ability for a user to rename an existing session file from within the Session Manager
- The ability for the user to enter a grid size upon loading a data set, which the program would use to construct a custom-sized grid
- Color picker GUI controls that would allow the user to select colors for the heat map and individual mice
- Pause functionality for animations
- The ability for a user to specify custom image / animation export dimensions

Despite these missing features, the program is still able to generate high-quality visualizations that provide valuable insight into individual and aggregate behaviors of mice from the experiment data. Once completed, the features listed above will add greatly to the overall user experience and visualization generation by providing greater user control and program feedback to the user.

Bugs in the system:

The program is able to compile and run without producing errors during the execution of its functions through most control paths. However, several bugs persist that my team was not able to fix by the program submission deadline. These bugs include:

- Issues that result from loading two or more data set files in the same session (from the time the user opens the program to the time the user closes the program). Noticeable effects from this bug include incorrectly generated visualizations and possible runtime exceptions during visualization generation. This was a bug that was reported by our test team. We attempted to fix this by resetting the state of our Mice and Grid objects in the openFile(File file) function of the AppStageController.java file, which should flush the old data from the preceding data set upon loading a new one. We ran out of time attempting to fix this bug.

- There is an issue when generating animated vector and overlay maps at very high frame rates and/or very high numbers of frames. Noticeable effects include the program's user interface locking up and the visualization animation skipping frames. This is most likely due to the animated vector map algorithm, since this phenomena does not occur while generating animated heat maps under similar conditions. My guess is that there is a logic error causing massive memory consumption, which over time will overwhelm the thread that is created during animations. Since the animated vector map algorithm performs well at modest frame rates and within modest frame sizes, and due to time constraints, we did not attempt to fix this error.
- When loading a session file, the Stop index may not be restored in the stopDataRangeTextArea GUI control from its corresponding value in the session file. This could be due to an event handler attached to stopDataRangeTextArea that saves its text value to the current session's session file every time stopDataRangeTextArea's text is modified; this event handler is located in the updateSessionTextAreaContent() function in the AppStageController.java file. The openFile(File file) function of the AppStageController.java file also contains code that attempts to set a previous session's Stop value to the stopDataRangeTextArea object.
- The image export code takes a snapshot of a rectangular portion of the screen containing the visualization. Currently, if another window is dragged on top of the program's visualization area during an image or animation export, then the content of the window on top will be unintentionally rendered in the output of the export. This issue used to manifest itself in the export of single images; the exported file would contain a very faint overlay of the file explorer window used to save the image, which would be captured in the export just before it disappeared. To get around this error (as a temporary solution), we currently sleep the main application thread for 500 milliseconds during export operations in the exportAnimation() and exportImage() functions of the AppStageController.java; this enables the file explorer window to disappear completely and not impact the content of the export.

These are bugs that we were able to identify, but more may exist.

Flaws in the system's software engineering

Although my team was able to produce a working product within the development timeframe, there were a couple major sacrifices to the quality of the code that were made for the sake of time. The first issue concerns a disconnect between the methods an object is designed to perform vs. the abstract concept that its class represents. An example of where this problem occurs in the source code is the Grid class, which started as a class that simply contained an array of GridSectors and was capable of generating Canvas objects of visual data. Over time, animation capabilities were introduced into the system, but instead of creating an animation class to govern the Grid's rendering based on the visualization options, the Grid class assumed these animation responsibilities; as a result, the Grid class no longer represents just the experiment enclosure, but instead a majority of the program's animation and visualization components. For the sake of modularity and easier code maintenance, these components should be moved to new classes.

The second flaw, and most serious, is the lack of enforcement of the Model View Controller design pattern. The main reason for this flaw is the ignorance shared by my team and I of the MVC features in JavaFX, which is no excuse for not implementing proper MVC code. If my team could redo the project, we would spend several weeks practicing nothing but JavaFX's data binding and event listening code in order to properly architect the program's inner workings. The consequences of failing to enforce the MVC pattern are simple: it becomes necessary to manually set the state of GUI objects that change and it becomes much more difficult to manage these GUI objects over time. If the next development team were able to rework the source code in such a way that it adheres to the concepts of the MVC pattern (which is easier said than done), then the code would be more maintainable, better performing, less error-prone, and of a higher quality in general.