Psychic Waffle Project Proposal

A.J. Feather
af2849@columbia.edu

Andrew Grant
amg2215@columbia.edu

Jacob Graff
jag2302@columbia.edu

Jake Weissman
jdw2159@columbia.edu

December 3, 2016

1 Second Iteration Overview

2 Static Analysis

We used pylint to run static analysis on our python code base. According to their metric, we didn't score too well and had a lot of changes to make. However, the most common mistakes were lines too long, unnecessary parenthesis, and style changes along those lines. Pylint want also unhappy with our naming conventions. We made a serious effort to make all of the code PEP8 compliant and got a lot accomplished. We greatly reduced the number of issues that pylint complained about. The vase majority of the remaining errors were unused imports - sine we used wild card imports (from blank import *) pylint complained about a lot of unused imports. It would have been too long to track down exactly which functions we were using and from which imports so we decided to leave it. There were some other errors that we fixed in some files (such as spacing, naming, variable uses, etc.), but combing through all of the lint suggestions and making hundreds of fixes would have taken too long and we unfortunately didn't have time for it. We made most of the glaring changes and left some of the less sever code smells as they are, thereby prioritizing the more important changes given our limited time constraint. The results of our initial static analysis can be found here (https://github.com/andyg7/4156Project/blob/master/docs/combined-pylint.txt). In hindsight, it would have been better if we worked with a static analyzer from the beginning instead of at the end. This could have helped us develop with style guides more in mind and write much cleaner code. This is a glaring lesson from running this static analysis.

3 Code Inspection

One component that Jake and Andy worked on was the multiprocessing architecture to the project. The files that we examined were multi_processing_handler.py and server.py. Andy was the reader and Jake was the recorder. We took notes by hand but here is the summary and highlights of the discussion. One of the first things that Jacob noticed was the lack of object oriented design, more specifically both files were implemented without any classes. This made the code messy and hard to read. For example, in the multiproc handler code, it would have made a lot of sense to use a class because the queue was a parameter to each function. Furthermore, this made it hard to test. Another thing we noticed was the presence of magic constants. Specifically, the workload that the queue processed was a list, and it wasn't clear what this list truly consisted of. There weren't many comments and even worse there were outdated comments.

The component we analyzed for Jacob and A.J. was the UI based code (html files), specifically completed-list.html, active-list.html and server.py. Jacob was the reader and A.J. was the recorder. One of the first things we discussed was the existence of duplicated code in both html files. Both files performed similar functions and some of that could have been abstracted out. Furthermore, the code powering the UI in server.py was not a clean interface to the UI code and it was scattered throughout the python code. We

discussed ways to condense it and make it clearer how the python code was interacting with the UI. Further, the formatting on the html files were off, making the code difficult to read and edit the UI code by anyone other than the writer.