2XB3 LAB 1

F. YANG
400243777, yangf51
H. CHIU
400261400, chiuh6
Y. LUO
400254211, luoy94

CS L02
(Jan 18th 2020)

Method

Repository may be found here: github.com/frankyang3/2xb3lab01

Everyone began by creating their own named files, <name>.txt and pushed to the repository. No merge conflicts occured. Following that, Harrison created an empty code.py file, and pushed. Everyone else pulled and obtained an empty python file. Everyone built their own python function for are_valid_groups and pushed. The merge conflicts were fixed by taking one person's code, as everyone's algorithm was quite similar. Then Yi continued by adding a limit of 2-3 people per group, as well as unique student numbers and pushed. Finally everyone took turns as player vs. adversaries, changing a line or two of the code for the player to fix.

Analysis

The first push went smoothly, everyone could pull then push and see name.txt
The second push on the same file didn't work as easily. After one person pushed, we found that no one else could push. Each of us tried to pull the existing code, but found merge conflicts. We fixed these on at a time on one person's local, and pushed a working version to the remote. The final merge conflicts came at the end, with two adversaries vs. one player. This went smoothly as the adversaries ended up changing different portions, resulting in no merge conflicts. The player was easily able to resolve conflicts each time.

We have found a few types of merge conflicts, when two people write on the same file from scratch, when two people change different parts of the file, and when people add new files to the repo. The first is the most difficult to resolve, as you end up with entirely different files. The only way to merge that is by rewriting or rejecting one person's code entirely, erasing work. The second merge conflict I believe would be the most common, and is relatively easy to fix with communication. This requires one person to pull

and merge the changes locally, and as long as the local version works after the merge, all is well. The final case is where different people add new files. As long as there aren't duplicate file names, there should be no merge conflicts, making this the easiest case to resolve. Creating branches where each branch is for testing sections of code which are selectively merged is a great way to prevent merge conflicts.

Conclusion

We learned that it is extremely important to communicate with one another to ensure that each of us understands the what and where the changes the others are making. Communication reduces the amount of merge conflicts.

When people work on the exact same function but make drastic changes in the logic inside it, it is easy to change because we would only need to choose which function we would want to keep. If many people create many large changes that affect the entire file, it is difficult to change because then there would be a lot of merge conflicts. Merge conflicts have to be resolved manually which requires the analysis of the code to compare which changes should be kept or discarded. It is also difficult because there are many changes to look after which is why constantly pushing even for small changes are necessary. In this way, the people working on the code will always have the most updated and same version that everyone is working on.