Team 147

CSE 116

May 6, 2016

Dr. Carl Alphonce

Stage 3 Code Review

Deciding on the codebase for Stage 3 was not a difficult decision, as we wanted to pick the codebase that most closely resembled our Stage 2 codebase. The first candidate, Candidate A, AKA DragonTeam, was the first candidate that we nixed. Since we are required to have complete javadocs for this stage no matter the candidate chosen, Candidate A was off the short list. There is very little documentation, and what for what documentation there is, it doesn't fully explain how their code functions--at all. It gives very general ideas about the methods, but the methods are so convoluted and messy that we couldn't grasp what was happening in the code. Not only that, the documentation is only line comments, not javadocs. Already, our workload would be massive in just fixing up the javadocs.

There is no class for a player object, as they are represented internally in the Board class as a two-element integer array. We note that this code feels a lot like a procedural undertaking than an object-oriented one. Also, the Tile class does not have any documentation as to what element in the open paths array represents what path, ie north, south, east, or west, on a Tile object. As such, the modularity of this candidate is, well, abysmal. Adding in the new functionalities wouldn't be easy and keeping track of the code without refactoring a lot of it would be difficult. As for tests, there are quite a few of them and only four fail, but we want a

thoroughly tested codebase where all the tests pass. The code itself is actually very well organized, visually, which would is certainly a strength of this codebase, but without any documentation this code is completely alien to us.

When we played the game, there is an obvious bug where if you click on a tile to move to, the pawn will sometimes jump to a completely different, unrelated tile. Also, there is a size button that does literally nothing good. It just makes the window larger and larger for no reason, and since we're not allowed to delete code that's not broken (the button does work as intended), we refuse to work on this codebase. Candidate A is thus rejected.

Candidate B, AKA MasterLabyrinth, proved to be a more robust codebase, but ultimately fell short because of problems similar to Candidate A. The codebase has multiple objects, including a much desired Pawn class. Adding the new Stage 3 functionalities wouldn't be very difficult in this codebase. The variable names and parameter names were adequately named so that we could get an idea of how the code works through their naming schemes. The GUI components, though, were extremely disorganized with weird indentation and obscure method calls.

Adding on to that, the documentation is sporadic at best and would have been helpful for us when playing through their game. The interface is completely unwieldy, navigating the gameboard is absolutely foreign and unintuitive. There is also a lot of missing information in the GUI that should be displayed. Unit tests were good, and they all passed, and it looks like they did test very thoroughly the functionality of their codebase If we knew how to work the interface, perhaps we would have chosen Candidate B, but since the code is just too poorly documented, we rejected the codebase.

Candidate C, AKA CSE Group, had good modularity with its different object types and had very well documented code. Not only is it well documented, but it's documented in the javadoc format. It's incredible that two of the three codebases we've looked at so far have not had correct javadocs. Regardless, this code is horrifyingly messy and has unnecessarily convoluted method definitions that greatly reduce the clarity of the code. Also, there is missing functionality in the code. Multiple players can't be seen on the same tile, and tokens are automatically picked up (which they should not be). The GUI doesn't have all the required information on it either, and the visuals of the game, albeit pretty, are very hard to see and understand.

There are several instances in the code where there are massive if-else selection structures that could be made into switch statements or nested for loops. As nice as it is having multiple classes, the modularity of the code is not great and would not mesh well with our team's coding style. Moreover, the unit tests that they have are barely passing at all. A vast majority of them fail or have errors and the tests don't really say why they're failing. More than anything else, we rejected Candidate C because of poor testing.

Candidate D, AKA Master Labyrinth, is rejected outright. This codebase is awful. When playing the game, there is no checking for arguments, and if you overlap players, the game deletes the player that was on the tile first. It's not a visual bug, it's an internal bug as the player that was first on the tile is deleted from the game, and the game then throws a NullPointerException. For Stage 2, this is unacceptable for our team's standards.

The code itself is very poorly organized. The Board and GUI classes have over one thousand lines of code, and the rest of the classes have barely any documentation and code.

Since so much of the game logic is handled inside Board, modularity is practically nonexistent. Adding new functionalities to this codebase would be incredibly difficult and take too much time to implement in the time that we have for this stage. As for tests, there is only one test class, and many of the tests fail. Looking at the test code, the tests all look very weak and poorly designed. This codebase is firmly rejected.

Since there are no other codebases to reject, we choose Candidate E. Candidate E has a very robust user interface, that shows all the required information. It's easy to use and understand what is happening with the GUI. The code is well documented, almost overly documented, and it's in javadoc format, too. There are a multitude of classes that are aptly named, and the instance variables and method names are also well stated. Modularity is definitely good, and it looks very simple to get the required information out of the game for the save file, and it looks easy to add in the new code that we need for this stage.

However, some of the code has poor documentation, but the poorly documented parts are very easy to understand by looking at the code. Code clarity is certainly the best out of all of the candidates. The only bug we found was that you can delete the text off of the tokens, but that fix is very easy, as it's just setting the text field to not be editable.

The unit tests are insane. There are over two thousand tests, and they test many different circumstances that come up in the game. They're broken up into different classes in different test packages, so they're really easy to add to. Moreover, the codebase is by far and away the closest in implementation to our Stage 2 codebase.

All things considered, Team 147 chooses Candidate E for its Stage 3 codebase.