CSE 116 - Spring 2016 - Dr. Alphonce
Team 159
Ian Wilson, Satya Kranthi Penumanchili, Josh Maniarasu, Keming Kuang, Weijin Zhu
05-06-16

Candidate A:

Initially we thought Candidate A had potential due to the fact that the implementation of the game's GUI looked far better than the rest of the candidates.However after actual analysis of the code base we decided to choose a different candidate due to numerous problems. The strengths of the code base is that the GUI looked great compared to the other code bases and the structure of the code is properly laid out.The GUI is visually pleasing to the eye which makes it appealing to the player. The code is laid out in two packages named "code" and "gui".Testing is done in two other packages called "boardTest" and "TileTests". The code base has separated game mechanics and visual implementation of the game which allows the creator to edit the game without have to change a lot and also allows for modularity.

The weaknesses of Candidates A's code base completely outweigh its strengths even though the structure of the code is good. The way the game is implemented is inefficient. The GUI is made by adding multiple layers on a JFrame and is updated by adding more layers on top. When a change is made to the game a new layer is added to JFrame and the previous layers aren't deleted. Due to not deleting the previous layers it cause the game to take up more space each time a change is made. It also creates a lot of new objects every time a tile is inserted, which causes problems in tracking objects that exist in the game, and uses more memory that it needs to. While the GUI is visually pleasing, the readability of the code is moderately difficult. The names of the methods do not precisely correspond to what they actually do. For example, they have a method called "eat" which is actually a method for the current token. "Eat" might make sense for an animal class, but for a token, this naming scheme is questionable. The code also has minimal documentation that isn't properly done. Instead of Javadocs, they used plain comments. In general, the code appears "long-winded", using many lines to do what could be accomplished more succinctly (but, we admit that our code was also a bit guilty of this for stages 1 and 2). When the size button is clicked the JFrame of the board is made bigger to properly adjust JFrame to different resolutions. Although it is a nice feature, more problems could have been dealt with rather than adding functionality that isn't demanded by an employer. While having a decent amount of tests done they didn't cover all the functionality that needed to be tested. Out of the seventy tests, twenty-three of them that did not pass. In addition, it missed addressing key logical rules such as picking up tokens only if the player wants to and the game doesn't end when the token number 25 is picked up. Overall the complex/bulky code of Candidate A makes it hard to add new functionality.

Candidate B:

One of the things our team liked the most about candidate B was the way this group set the fixed tiles up in Board class. The authors used boolean values to set the directions for each tile. The authors also show us the coordinates of each tile, which gave us a better understanding about how the game board is set up. There is also a good amount of Java documentation, properly Javadoc-ed.

A primary weakness of this group's code is that the code is really difficult to follow. We had tough time understanding some of the code, because some of the variable names do not correspond to functionality, and when these variables are used over and over, it caused understanding what's going on more difficult. Another problem we noticed was that the authors randomize the shape of the tiles without restriction to how connecting paths a tile can have; therefore besides only L-Tiles, I-Tiles, and T-Tiles, there are also sometimes (when randomly generated in that instance) tiles with a "cross" shape (four connecting paths on the tile). The game also was not designed with the restriction that you need two to four players inclusive; for example, when we tried to run the game with one player or with more than four players, the game starts, despite this being against the game rules. Yet another problem is that when players insert the extra tile from one side, the next player is able to insert the tile from the opposite side, which violates the game rules. The authors also used Java applets to make the game, something that none of our team have used before, so this would make it hard to add functionality for us in particular. Since none of are familiar with Java applets this is one of the primary reasons our team decided against using candidate B (in addition to the other problems we mentioned).

Candidate C:

The design of the code was well done and separated into multiple packages, which allows for us to understand what the code does more easily because the functionality is clearly delineated. In addition, this allows for the easy addition of new functionality as well. They also have "code.GUI" which holds all the listeners for the package"GUI" to use. Overall this makes it easier to work on Candidate C's code base. We also found the code fairly easy to understand, which was a primary motivator behind us choosing candidate C. The code is clean, well spaced/laid out and has proper names for the methods. The code base is well documented, with many classes and method having proper Javadocs, and overall had more proper formatting compared to the other candidates, which aids in readability. The GUI looked clean and neat, with most of the functionality working, like inserting tiles and having a rotate button. Overall the

strengths of Candidate C's code base were quite high in our opinion. Code C is the candidate we chose.

Despite all of these strengths, there were also problems that we found after playing around with Candidate C's code more. The issues were mainly not adding all functionality required for stage 2. Some of the problems were logical in nature; for example, the insert extra tile functionality works well, but a problem is that a player can insert the extra tile at the same position as where the extra tile had left the board on the previous turn--a violation of the game rules. The game was also only designed for four players, and does not accommodate two or three. However these issues didn't look too hard to fix since the code base was similar to our own code base that we worked on for stage 2. Another problem we noticed was that of the fifty-one JUnit tests present, there were 16 errors and two failures, leading us to suspect there were major problems with the code. As it turns out, these problems were easily fixable. We also thought that the number of JUnit tests present was very low, but after reading them it seemed that they had decent code coverage. Overall the code of Candidate C looked clean and had good readability which allowed us to easily implement new functionality into the code.

Candidate D:

One of the strengths of the Candidate D's code is documentation except for in the class GUI, where documentation was omitted. It has Javadocs on top of each method which helps to understand the code. Another strength is the authors of this group named each method in such a way that the functionality of method is reflected in it's name; for example one of the methods is named "createTokenTiles", as what it says, this method creates the tiles, on top of which, the tokens will be placed when the game begins.

At first glance we thought the presence of javadocs was really great, but as we read into what the javadocs said, we realized that they were not very clearly specifying what those classes and methods do, which makes it tough to follow the code. One of the biggest weaknesses we found, are that there are many logical errors present. One of them is that the game is designed for 0 to 4 players, while it should only be designed for 2, 3 or 4. There are also many unused methods. When the game starts, we also noticed that the GUI displays that each player already has 9 points. While this is true based on the number of wands that each player has left (3 each at the beginning of a game), we are only supposed to calculate a score at the end of the game, as per the game rules. Other problems include that pawns overlap on the GUI, and also a player can still play after picking up tokens. When a player tries to insert a tile and coincidently if there is a pawn on the opposite side, this will make the pawn switch to the tile outside the board (the extra tile), which means if one inserts the tile to shift the board, the pawn will appear back on the

board and on the tile that is inserted. When it comes to the JUnit tests, there are a total of 54 tests, but only 8 of them pass; this does not give us high confidence to take up this code base.

Candidate E:

Our team (team 159) created the code that is candidate E, therefore (and by instruction of Dr. Alphonce), we have not reviewed this code base.

CONCLUSION:

Overall we decided to choose Candidate C's code base because of its easy readability, our understanding of the code, and our feeling that implementing new functionality for us would not be very difficult. Some of the ways that this group went about their implementations was similar to what we had done previously in stages 1 and 2. The design of the code base is well thought out and implemented. The javadocs were useful to understanding the code base. The issues it has were less problematic compared to the other code bases, in our opinion.