Assignment 3: Code Review Report
Group 19 - Arda Cifci, Nayeong Lee

The first code smell we identified in our code review was how we were making labels (text boxes) in our UI. In each screen we were using the same 9 lines of code several times to create each label with the only difference being the text itself and its position on the screen. This needed refactoring as it was a case of duplicate copy-and-pasted code which ultimately bloated our classes and their corresponding methods with unneeded duplicate code. Our solution to this code duplication problem was to move the creation process of a label into its own separate class and just pass the needed parameters into the label object's constructor to create a new label. For this refactoring process, we first created a new class called Label and moved over the lines of code required to make a label into its constructor. The constructor also takes in all the required specific inputs that are needed to create the unique label (all of this can be seen in Commit c309cf10 near the bottom). After creating this class all we had to do was replace all the existing duplicate 9 lines of label code pairs with 1 create label line (Commits c309cf10 and 0583b929 near the bottom). By using this solution, we got rid of this code smell and future proofed our label making process.

The second smell we identified was the enormous amount of unreadable if-else statements in our singular enemy player tracking method "fryCookTrackingPlayerMove" in the gameloop class. This method was the result of our group not being able to implement a proper path finding algorithm like A* and instead settling on our own implementation from scratch (can see this implementation in commit 1f017ab7**)**. Our solution to this code smell was to turn this method into its own separate class and break down the many if-else statements into various meaningful methods. For the refactoring process, we created the PlayerTracker class and moved the method "fryCookTrackingPlayerMove" into it (commit 1f017ab7**)**. There we took sections of the if-else statements and moved them into their own meaningful methods. Inside these methods we further decomposed the code into more methods as many of the new methods had duplicate code inside them, or were still complicated and warranted more methods (this can be seen in commit 9cc0a5de, expand the PlayerTracker.java section). Additionally, many of the statements in the "fryCookTrackingPlayerMove" method used many random int primitives to decide logic. Instead of that, during the decomposition we changed most of these int primitives into our already implemented constant enums which removed a lot of the "magic number logic" of the code. By doing these refactors, we were able to remove a large portion of code out of an already busy class, and decompose it into a class that is more understandable and reusable.

The third smell we identified was the several instances of code duplication in the gameloop class. Whenever we needed to reset the game's state or display the lose screen of the game, we would reuse the same copy and pasted code. Our solution to this code smell was to place these two sets of codes into their own methods. For this refactoring process we created the two methods, resetEntityAndKey and openLoseScreen, and moved the respective code into each one. Then we went through the gameloop class and replaced all the existing duplicate codes with a call to this method (These can be seen in commits 8d8e544d and b46fe2f0). By doing this refactoring, we removed many lines of duplicate code and future proofed resetting the game's state and opening of the losescreen from the gameloop class if we were to ever need them again.

The fourth code smell we identified was that we included the whole creation of buttons in each of UI screens(DifficultyScreen, HelpScreen, etc). Similar to the first smell mentioned above, in each class, we would have 7 lines of code just for one button creation and attributes. (ex: Commit bf038023, line 115-121)If each UI

screen has more than one button, and there are six different UI screens, the amount of lines for setting button attributes take up a huge part of the code. These lines can seem very redundant especially since the button fonts, backgrounds, etc remain the same for all buttons. We thought this would be a good place to refactor since it was an example of “code duplication” smell. To solve this code duplication smell, we can make a separate class called “button” where it is dedicated to creating a button and setting its attributes. Instead of just copy and paste the same structure of button creation codes in each screen, we can now just pass the parameters needed into the button constructors to create buttons. This itself has reduced at least 7 lines of codes in each screen. We refactored by creating a new class “Button.java” and moved the lines of code needed to make a button into its constructor. Such lines of code include setText, setBackground, setFont, etc. (The whole implementation is in Commit c6ab19de). After making this class, we simply replaced all redundant 7 lines of button codes with 1 line of “new Button” code (ex: back = new Button("BACK", this, 1024, 200, 200, 100); from Commit c6ab19de). We did this replacement process for all six UI screens.

The fifth code smell we identified was that for each UI screen, we have multiple implementations for the actionPerformed() method. For each button, we had an actionPerformed() method that toggled that button. For example, in Home Screen (Commit 47798a27), we have three implementations for the same actionPerformed() method(line 93, 120, 141), except they toggle different buttons. The same problem can be found in other screens too. This is an example of “code duplication”. In fact, it is not necessary to have the same implementations when we can join them all together. To solve this problem, we came up with an idea to simplify the codes by putting all codes into one if/else statement. For instance, instead of calling the method 3 times for HELP, START, and EXIT buttons, we can call the method once where if (pressed.equals(“HELP”)), then toggle HELP button, else if (pressed.equals(“START”), then toggle START button, and so on. This not only helps the viewers to understand better, but it also reduces the codes by a lot. We did the same procedure for other screens. We refactored redundant actionPerformed() calls by putting them all in if/else statement. For LoseScreen (Commit f4bd7104), refactoring can be found in line 104-142. For PauseScreen (Commit 8beb60ee)**,** the refactoring is shown in line 110-137. Relevant commits also include e092e27e(gameloop), 14d5c829(winscreen), bf038023(helpscreen)

The sixth code smell we identified was that the UIcreator.java only consisted of one giant constructor where we threw all our initial UI creation code, such as creating the jframe and different jpanels, into. (Commit fb1d34c5) The main problem was that the constructor was too big with many different calls and the class was poorly structured which can easily confuse readers. The constructor itself took up almost 50 lines. We came up with a refactoring solution to make a separate method for each, and place them outside the constructor. After making all panel objects, instead of putting all the other methods all in one big constructor, we can take out all getters, setters, setBounds code, and add code from the constructor and make separate methods for each.(These three method declarations are in line 44, 56, and 67 respectively.) This way, we can simply just call that method in the constructor, which will make the code more concise. For the refactoring process, we used the “Extract Method” feature, and we named each method “private static void addPanels(), private static void setBounds(), private static void getterSetter()” respectively. By doing “Extract Method”, the constructor now only has three method calls (ex: getterSetter(helpScreen, game, pause, diffscreen, title, win, lose)). And the naming of each method is straightforward for readers.

Throughout the whole refactoring process, we also continuously looked for dead codes and unused variables and removed them.