Assignment 3
Group 14
Gabriel Cheng, Misha Goldenberg.

The first code refactoring we completed is renaming bad variable names, we followed a lower camel case naming convention however some variables had been named using underscores, so we renamed all the variables to follow the same style, camelCase. The commit that represents these changes is “Refactoring to camelCase 967ac1d1”

```
-   protected HashMap<String, BufferedImage> squirrel_pngs;
-   protected HashMap<String, BufferedImage> hidden_squirrel_pngs;
-   private BufferedImage chocolate_png;
-   private HashMap<String, BufferedImage> bear_pngs;
-   private BufferedImage bush_png;
-   private BufferedImage hunter_png;
-   private BufferedImage peanuts_png;
-   private BufferedImage[] tree_pngs;
-   private BufferedImage board_png;
-   private BufferedImage exit_png;
-   private BufferedImage trap_png;
-   private BufferedImage button_png;
```

```
+   protected HashMap<String, BufferedImage> squirrelPngs;
+   protected HashMap<String, BufferedImage> hiddenSquirrelPngs;
+   private BufferedImage chocolatePng;
+   private HashMap<String, BufferedImage> bearPngs;
+   private BufferedImage bushPng;
+   private BufferedImage peanutsPng;
+   private BufferedImage[] treePngs;
+   private BufferedImage boardPng;
+   private BufferedImage exitPng;
+   private BufferedImage trapPng;
+   private BufferedImage buttonPng;
```

Secondly, we noticed a lot of switch statements regarding the game's difficulty settings and the random map generation. For example, there was a switch statement that took in the game difficulty and return 5, 10, 15, and 15 to represent how many rewards to spawn for the easy, medium, hard and infinite difficulties. We refactored these switch statements by implementing getters into the difficulty enumeration. And used the @Override feature to specify the return value for the specific difficulties. This was done for the enemy count, reward count, trap count, etc. The related commit is “refactored difficulty switch statements 7e50c79a”

```
switch(dif){
    case EASY: minProximity = 5;
        break;
    case MEDIUM: minProximity = 3;
        break;
    case HARD:
    case INFINITE: minProximity = 0;
        break;
}
```

```
minProximity = dif.getMinRewardProx();
```

```
public enum Difficulty {
    EASY {
        @Override
        public int getTrapCount() {return 4;}
        @Override
        public int getMinRewardProx() {return 5;}
        @Override
        public int getRewardCount() {return 5;}
        @Override
        public int getEnemyCount() {return 1;}
        @Override
        public int getMinEnemyProx() {return 10;}
        @Override
        public int getBonusRewardCount() {return 3;}
    },
```

The third refactor that we completed was poorly structured code, all the display name variables were just contained with title values of 1..7. This did not allow for the code to be easily understood. So we changed the code to contain more descriptive title values like “main”, “settings” etc. The commit is “refactored display names 08e43195”. In this commit, we also optimized some poorly structured map generation code by reducing the number of loops from 4 to 2.

```
-        displayPanel.add(titlePanel, "1");
-        displayPanel.add(playPanel, "2");
-        displayPanel.add(settPanel, "3");
-        displayPanel.add(diffPanel, "4");
-        displayPanel.add(pausePanel, "5");
-        displayPanel.add(gameOver, "6");
-        displayPanel.add(gameWon, "7");
```

```
123 +        displayPanel.add(titlePanel, "Title");
124 +        displayPanel.add(playPanel, "Play");
125 +        displayPanel.add(settPanel, "Settings");
126 +        displayPanel.add(diffPanel, "Difficulty");
127 +        displayPanel.add(pausePanel, "Pause");
128 +        displayPanel.add(gameOver, "GameOver");
129 +        displayPanel.add(gameWon, "GameWon");
```

The fourth refactoring that we completed was for poor code scalability and a large case statement and refactoring regarding drawing all the graphics. The switch statement would receive the type of value to draw at a specific X and Y coordinate and use a specific method to draw it. Now we added general draw() methods to the Objects enumeration. The objects that need a more specific drawing method will override the generic method. The map generation for static images now just calls the polymorphic .draw() method on all the Objects to draw the game board. The commit related to these changes is “Refactor object drawn pngs 9c13fe09”.

```
}else if (boardMap[col][row] != Objects.HERO && boardMap[col][row] != Objects.ENEMY){
    Object obj = boardMap[col][row];
    BufferedImage png = staticImages.get(obj.toString());
    boardMap[col][row].draw(g2, col, row, png, tileHeight, tileWidth);
}
```

```
-                    }catch(Exception e){};
-                    currentTree++;
-                break;
-                case HEROHIDDEN:
-                    if(!hero.isAtBush()){
-                        g2.drawImage(bushPng, col * tileWidth + 10, row * tileHeight+60 + 10, (int)
(tileWidth*(2.0/3)), (int)(tileHeight*(2.0/3)), null);
-                    }
-                    break;
-                case ENEMYANDBUSH:
-                case BUSH:
-                    g2.drawImage(bushPng, col * tileWidth + 10, row * tileHeight+60 + 10, (int)
(tileWidth*(2.0/3)), (int)(tileHeight*(2.0/3)), null);
-                    break;
-                case ENEMYANDTRAP:
-                case TRAP:
-                    g2.drawImage(trapPng, col * tileWidth+20, row * tileHeight+60+20, (int)
(tileWidth*(1.0/3)), (int)(tileHeight*(1.0/3)), null);
-                    break;
-                case ENEMYANDREWARD:
-                case REWARD:
-                    g2.drawImage(peanutsPng, col * tileWidth + 15, row * tileHeight+60 + 15,
tileWidth/2, tileHeight/2, null);
-
-                break;
-                case BONUS:
-                    //hide the object on the first render
-                    if(!firstRender) {
-                        g2.drawImage(chocolatePng, col * tileWidth, row * tileHeight + 60,
tileWidth/2, tileHeight/2, null);
-                    }
-                break;

-                //no exit image yet
-                case EXIT: g2.drawImage(exitPng, col * tileWidth, row * tileHeight+60, tileWidth,
tileHeight, null);
-                break;
-
```

The fifth refactoring we completed was for unsafe code. We did not specifically check for a null response when obtaining an asset in our code so we added non-null requirements when we obtain all of our different assets in the code. This was done in the “Added null safety, finalized variables 7eb93b1f” commit.

```
-            AudioInputStream musicstream =
AudioSystem.getAudioInputStream(getClass().getResourceAsStream("/fail.wav"));
```

```
37 +            AudioInputStream musicstream =
AudioSystem.getAudioInputStream(Objects.requireNonNull(getClass().getResourceAsStream("/fail.wav")));
```

The final refactoring that we completed was related to dead code, unused variables, and bad variable types. We walked through all of the files in our code and removed all the unused variables, and dead or commented code we could find, added object safe equals statements and we modified a lot of variables to either contain a final or to be instantiated as a local variable in the constructor for minor optimizations. All of this was done in the “Added null safety, finalized variables 7eb93b1f” commit.

```
-        /*
-        gameoverButton = new JButton("Test game over");
-        gbc.gridx=0;
-        gbc.gridy=6;
-        gameoverButton.setFocusable(false);
-        this.add(gameoverButton, gbc);
-         */
```

```
if(dir == "North" || dir == "West"){
```

```
372 +        if(java.util.Objects.equals(dir, "North") || java.util.Objects.equals(dir, "West")){
```