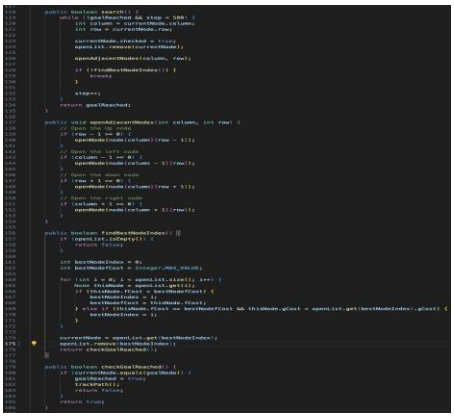
Assignment 4 Report

Ryan Martin 301445106
Jea Oh Lee 301351043

For this assignment we worked together to refactor code that we implemented during this project. In this report we will document the smells we have identified, starting with methods that are too long and that could benefit from being refactored.

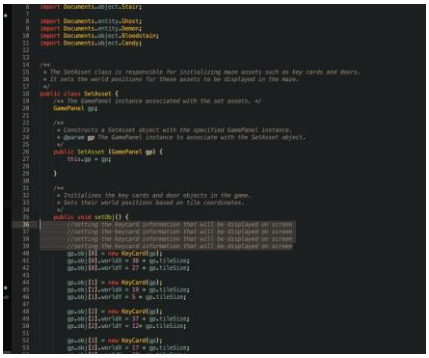
Starting with the bad smell of having methods that are too long, as this causes our code to have poor readability and maintainability. We analyzed our code for these methods and found three; search(), removeNode(), and setNode() in Pathfinding.java. We partitioned these long methods into smaller, more manageable functions, each with a clear and distinct responsibility, to increase readability and maintainability. Here is how the search() method looked before and after the refactoring. We applied similar tactics to the resetNodes() and setNode() methods.



As can be seen in the refactored photo (right), there was a general lack of documentation in our Entity and Pathfinding classes. We eliminated the "lack of documentation" bad smell by implementing javadocs explaining our methods to enhance understanding and provide clarity on their intended purposes. Both classes required extensive comments both in terms of variable and method uses.

Another bad smell we encountered was that of code duplication. The Monster, Entity, and MainCharacter classes all had the spriteNum implementation in their update() methods. The sprite number is used to determine which image will be displayed when animating our character's movement. We implemented a Utility Function to replace the handling and image loading since it is repetitive and could be abstracted into separate methods using our utility class. We implemented the utility tools in order to manage all the Bufferedimage scaling as well as the animation control. Since our entities and the map required image import, we decided to create a functional class that deals with only image related methods to reduce the duplication of the codes.

Throughout our refactoring we noticed that there were variables and imports that were not being used in each class.



```
import Documents.object.Stair;

import Documents.entity.Ghost;
import Documents.entity.Demon;
import Documents.object.BloodObtain;
import Documents.object.Candy;


/**
 * The SetAsset class is responsible for initializing maze assets such as key cards and doors.
 * It sets the world positions for these assets to be displayed in the maze.
 */
public class SetAsset {
    /** The GamePanel instance associated with the set assets. */
    GamePanel gp;

    /**
     * Constructs a SetAsset object with the specified GamePanel instance.
     * @param gp The GamePanel instance to associate with the SetAsset object.
     */
    public SetAsset (GamePanel gp) {
        this.gp = gp;
    }

    /**
     * Initializes the key cards and door objects in the game.
     * Sets their world positions based on tile coordinates.
     */
    public void setObj() {
        //setting the keycard information that will be displayed on screen
        //setting the keycard information that will be displayed on screen
        //setting the keycard information that will be displayed on screen
        //setting the keycard information that will be displayed on screen
        gp.obj[0] = new KeyCard(gp);
        gp.obj[0].worldX = 38 * gp.tileSize;
        gp.obj[0].worldY = 27 * gp.tileSize;

        gp.obj[1] = new KeyCard(gp);
        gp.obj[1].worldX = 19 * gp.tileSize;
        gp.obj[1].worldY = 5 * gp.tileSize;

        gp.obj[2] = new KeyCard(gp);
        gp.obj[2].worldX = 37 * gp.tileSize;
        gp.obj[2].worldY = 12* gp.tileSize;

        gp.obj[3] = new KeyCard(gp);
        gp.obj[3].worldX = 17 * gp.tileSize;
```

```
public void speak(){
    if(dialogues[dialogueIndex] == null){
        dialogueIndex = 0;
    }
    gp.ui.currentDialogue = dialogues[dialogueIndex];
    dialogueIndex++;

    switch(gp.mainCharacter.direction){
        case "up":
            direction = "down";
            break;
        case "down":
            direction = "up";
            break;
        case "left":
            direction = "right";
            break;
        case "right":
            direction = "left";
            break;
    }
}
```

List of unused variables/comments we found were:

1. randomI in Demon.java - the variable was declared but never used in the Demon class
2. actionLockCount in Entity.java - the variable was declared but never used in Entity class
3. test in Entity.java - the variable was declared for debugging purpose since it was not usage anymore we decided to remove
4. checkDrawTime in KeyControl.java - the variable was to check the drawTime in order to control the animation from the user input, however, we since UtilityTools.java mostly deals with the importation of the image, we decided to implement the animation on the UtilityTools.java instead of the keyControl.java itself. We were able to remove it after implementing the animation control in UtilityTools.java.
5. There were some unnecessary comments on SetAsset.java where it was taking 4 lines with repetition. We decided to remove the comment since the documentation (javadoc) was enough for us to explain what the method does in the code.

In our first phase UML, we wanted to create a NPC functionality where it may help the player to understand the overall goal and what each reward will do to the player. However, after we implemented the NPC, we figured that it was unnecessary to explain how to play the game using the NPC. Since we removed the NPC we no longer needed the speak() and dialogue array to keep the string for the dialogue.

While we were refactoring the image importation method in order to minimize the duplicate codes, we noticed the implementation has changed from our original implementation so it was causing test fail results when importing the image depending on the spriteNum. In order to fix this problem, we had to change the value of the expected value to ensure that the refactored version passes the test as we expect it to work.

Finally, upon looking through our game code, we noticed that there were variables that could be refactored by renaming them to better match their functionality, making them more descriptive and self-explanatory. This improves the readability of the code and aids in accurately representing what is intended to happen in our code. A few examples of this were: wxPos renamed to worldXPos, wyPos renamed to worldYPos, and vel renamed to velocity.