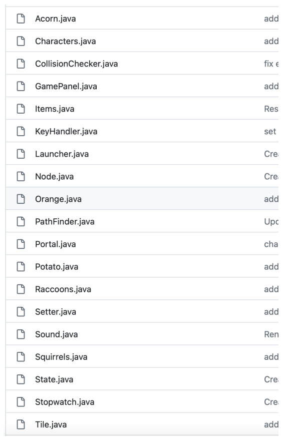
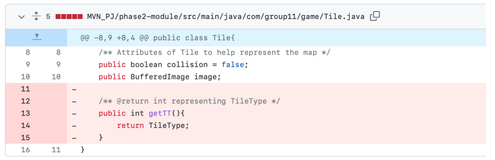


Acorn.java	add
Characters.java	add
CollisionChecker.java	fix e
GamePanel.java	add
Items.java	Res
KeyHandler.java	set
Launcher.java	Cre
Node.java	Cre
Orange.java	add
PathFinder.java	Upc
Portal.java	cha
Potato.java	add
Raccoons.java	add
Setter.java	add
Sound.java	Ren
Squirrels.java	add
State.java	Cre
Stopwatch.java	Cre
Tile.java	add

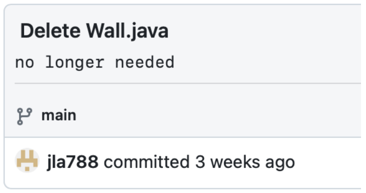
<u>Confusing class hierarchy:</u>

In the manual code review, one of the main things we came across when looking at the structure of our program was our confusing and ambiguous class hierarchy. When we take a look at the files, all of them are look like a cluttered spaghetti bowl (as seen below). Me and my partner implemented mainly the items class hierarchy (consisting items.java, Acorn.java, Orange.java, and Orange.java) which we then came up with a way to refactor this confusing class hierarchy. By changing the structure of the classes, we can refactor this to make it look much organized and less ambiguous with other classes in this spaghetti mess. However this process is long overdue, and in the attempt to do so many errors were encountered. A lesson learned to have organized architecture so the code can be organization and effectiveness.

<u>Dead code:</u>



```
5 MVN_PJ/phase2-module/src/main/java/com/group11/game/Tile.java
@@ -8,9 +8,4 @@ public class Tile{
         /** Attributes of Tile to help represent the map */
         public boolean collision = false;
         public BufferedImage image;
-
-        /** @return int representing TileType */
-        public int getTT(){
-            return TileType;
-        }
     }
```

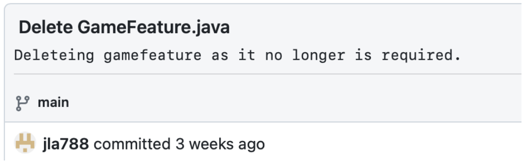


Delete Wall.java

```
no longer needed
```

main

jla788 committed 3 weeks ago



Delete GameFeature.java

```
Deleteing gamefeature as it no longer is required.
```

main

jla788 committed 3 weeks ago

Throughout the development of our game, that was many cases where dead code (even dead classes) were found and refactored. Although at this step of our implementation only one part of the game has dead code, we have refactored many bad code smells of this type throughout. Before I explain the previous smells, the current one is the import of java.util.random (in the class TileRepresenter.java), which was never used. Although this seems like a minor component, this is still mistake that needs to be addressed since it takes up memory in our game. Some of the biggest dead codes we had were Wall.java, GameFeature.java, and lots of unnecessary methods in the Tile.java class. These classes and methods were refactored by evaluating the necessity of the code, and then either deleting or implementing the code to good use.

<u>Low Cohesion:</u>

When taking a deeper look into the functionality of the classes we implemented, we saw some methods that have low cohesion in their classes and could use some refactoring. In the TileRepresenter.java class, we came across the getTileImage() and loadmap() methods that have low cohesion with the rest of our code. The purpose of the getTileImage() method acts as a setter, and should be implemented into the setters class, where the loadmap() method also does the same, and should be placed in the same class .

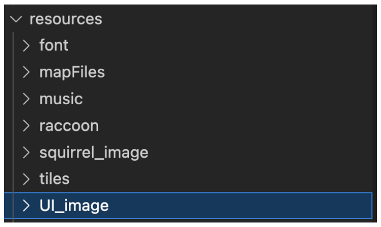
<u>Large class:</u>

The last code smell, and this one go hand and hand. The TileRepresenter.java class is too large for what its specifics should be. In order to fix this, similar refactoring from the previous statement are a good solution, as well as looking at other classes / making another class to manage the complexity of all the roles TileRepresenter.java is responsible for.

Badly Structure Project:

There were a couple of issues that are related to the code smell of a badly structured project. The naming of files and folders could be standardized and the images placed together for easier access and visibility.

The most notable example of this is how we structured the handling of images. While we did follow the standard maven practice of placing our code within


```
resources
  font
  mapFiles
  music
  raccoon
  squirrel_image
  tiles
  UI_image
```

src/main/java/com/group11/game

This main code pathing is fine, it's where we placed our resource folder that has issues While there is a folder for UI_image that contains most of our images. There are other folders created that contain images that can all be together instead of spread apart.

Bad/Confusing variable names:

There were variables called X and Y in tileRepresenter.java class and that was very confusing on what it does and what it represents. The variable names were changed to pixelX and pixelY so that it is clear that it is intended to set the size of the images.

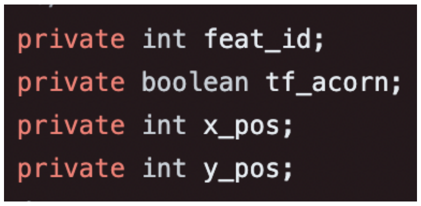
Lack of Documentation:

While going through the code there was a glaring issue with documentation. Specifically we lacked recorded specifications of the image sizes for certain elements of the game. We all assumed and verbally agreed that it would be 16 x 16 pixels but never had it recorded anywhere within our code.

This could lead to confusion within our games if there was a miscommunication as it would change the layout of the maps and be a glaring flaw within the project.

Unused Variables:

Throughout working on our project there were frequent times where unused variables or even variables that were taking up unnecessary memory that will never free. For example, there are a boolean called that never gets used. The fix we implemented is by having the variables removed and the overall class refactored into other classes. The issue lies with the fact there was poor planning in the structure of our project leading to variables being declared that served no actual purpose.


```java
private int feat_id;
private boolean tf_acorn;
private int x_pos;
private int y_pos;
```