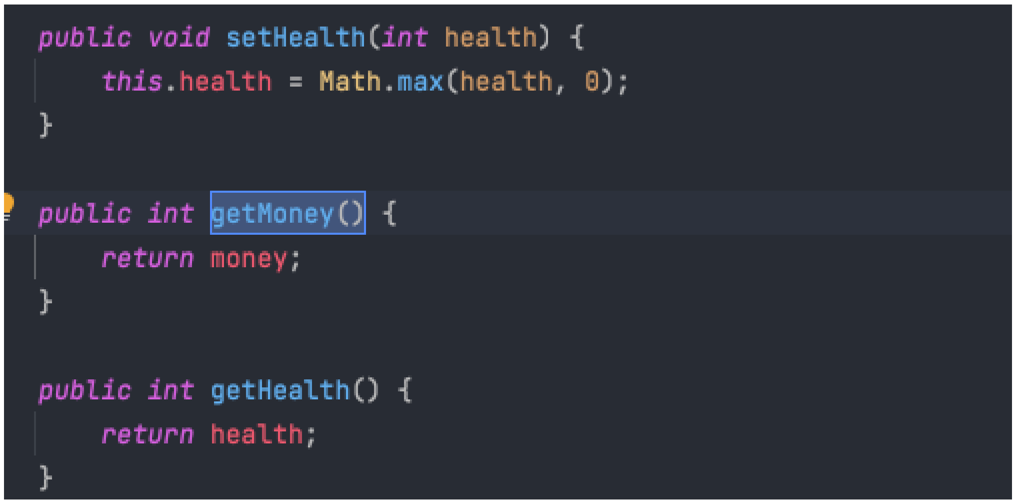
CS 2340

M5: Code Smells Write-Up

Code Smell # 1 - Dispensables; Speculative Generality

The method getMoney() is dispensable as it offers no other functionality outside or inside the class. The only time the money variable is accessed is within the class. For this purpose, it is unnecessary to have a getter method as it offers no functionality since the money variable is not accessed and does not need to be accessed within the Game Controller class. This code smell exists to support future features but has not been used so far.

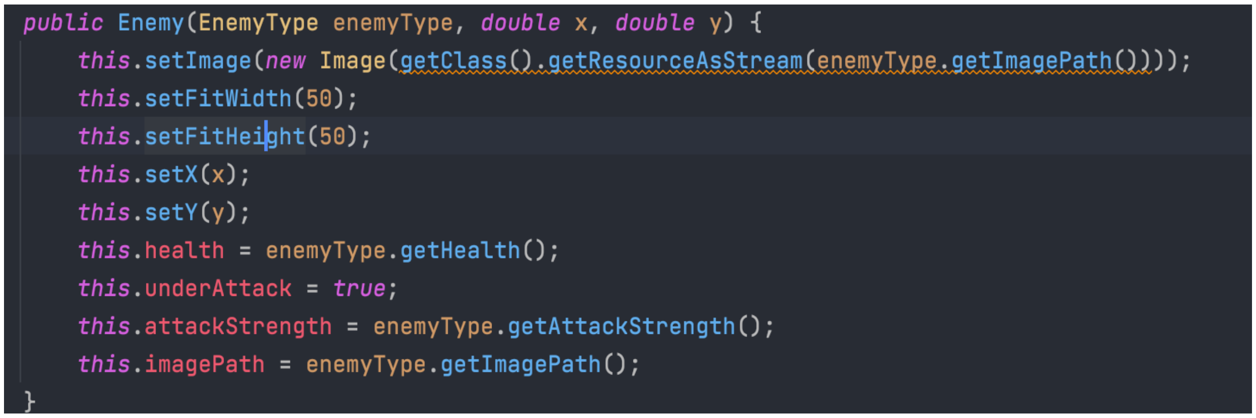


```java
public void setHealth(int health) {
    this.health = Math.max(health, 0);
}

public int getMoney() {
    return money;
}

public int getHealth() {
    return health;
}
```

To fix this, the getMoney() method was deleted. The code won’t be affected as the getMoney() had no clear purpose within the code.

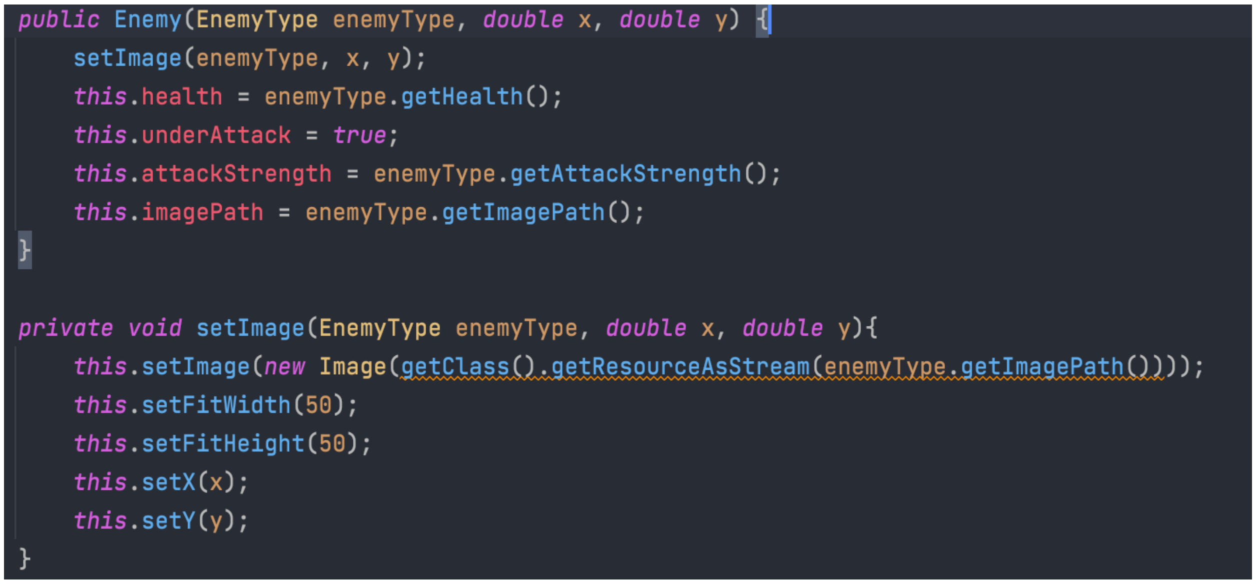
Code Smell # 2 - Bloaters; Long Method

The enemy constructor had some default functionality that was the same for creating every enemy constructor, and the constructor was getting quite long. The code smell exists as it was easier to group all of the enemy’s characteristics into the constructor eleven if it was getting quite long. Additionally, since all the elements within the constructor were referring to the enemy object, it was logical to group the characteristics.



```java
public Enemy(EnemyType enemyType, double x, double y) {
    this.setImage(new Image(getClass().getResourceAsStream(enemyType.getImagePath())));
    this.setFitWidth(50);
    this.setFitHeight(50);
    this.setX(x);
    this.setY(y);
    this.health = enemyType.getHealth();
    this.underAttack = true;
    this.attackStrength = enemyType.getAttackStrength();
    this.imagePath = enemyType.getImagePath();
}
```

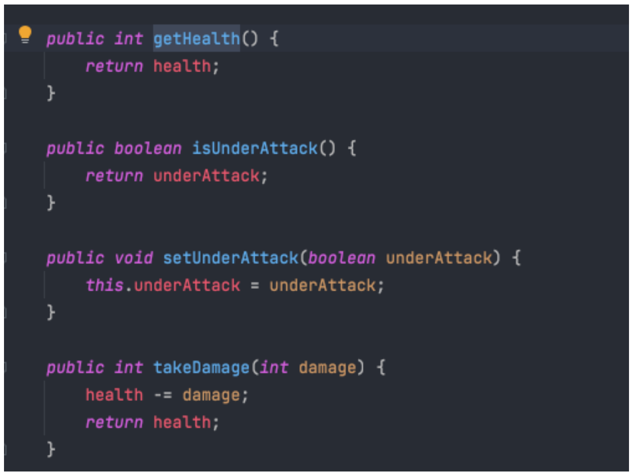
To fix this, a different method, setImage(), was created to create the enemy's image properly. This method made the image and set the image in the correct starting position. In this way, the constructor's size was reduced, and the code used to fix the enemy's image was put into one method.



```java
public Enemy(EnemyType enemyType, double x, double y) {
    setImage(enemyType, x, y);
    this.health = enemyType.getHealth();
    this.underAttack = true;
    this.attackStrength = enemyType.getAttackStrength();
    this.imagePath = enemyType.getImagePath();
}

private void setImage(EnemyType enemyType, double x, double y){
    this.setImage(new Image(getClass().getResourceAsStream(enemyType.getImagePath())));
    this.setFitWidth(50);
    this.setFitHeight(50);
    this.setX(x);
    this.setY(y);
}
```

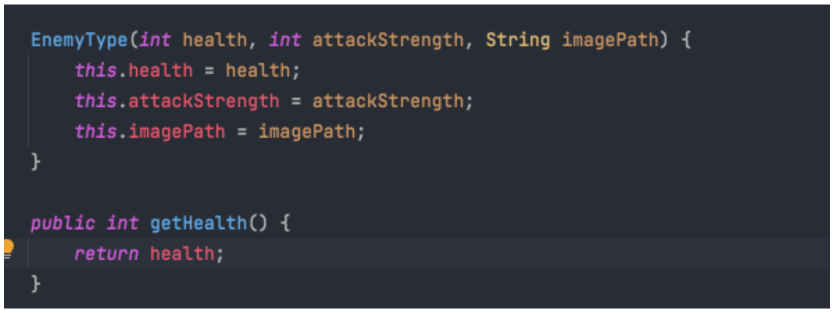
Code Smell # 3 - Dispensables; Duplicate Code

The method, getHealth() is dispensable as it offers no other functionality outside or inside the class. The only time the health variable is accessed is within the class. For this purpose, it is not needed to have a getter method as it offers no functionality since the getHealth() for the enemy is already under the Enemy Type class. This code smell exists to support future features, but it hasn't been used.



```java
public int getHealth() {
    return health;
}

public boolean isUnderAttack() {
    return underAttack;
}

public void setUnderAttack(boolean underAttack) {
    this.underAttack = underAttack;
}

public int takeDamage(int damage) {
    health -= damage;
    return health;
}
```

(a) Enemy Class



```java
EnemyType(int health, int attackStrength, String imagePath) {
    this.health = health;
    this.attackStrength = attackStrength;
    this.imagePath = imagePath;
}

public int getHealth() {
    return health;
}
```

(b) EnemyType Class

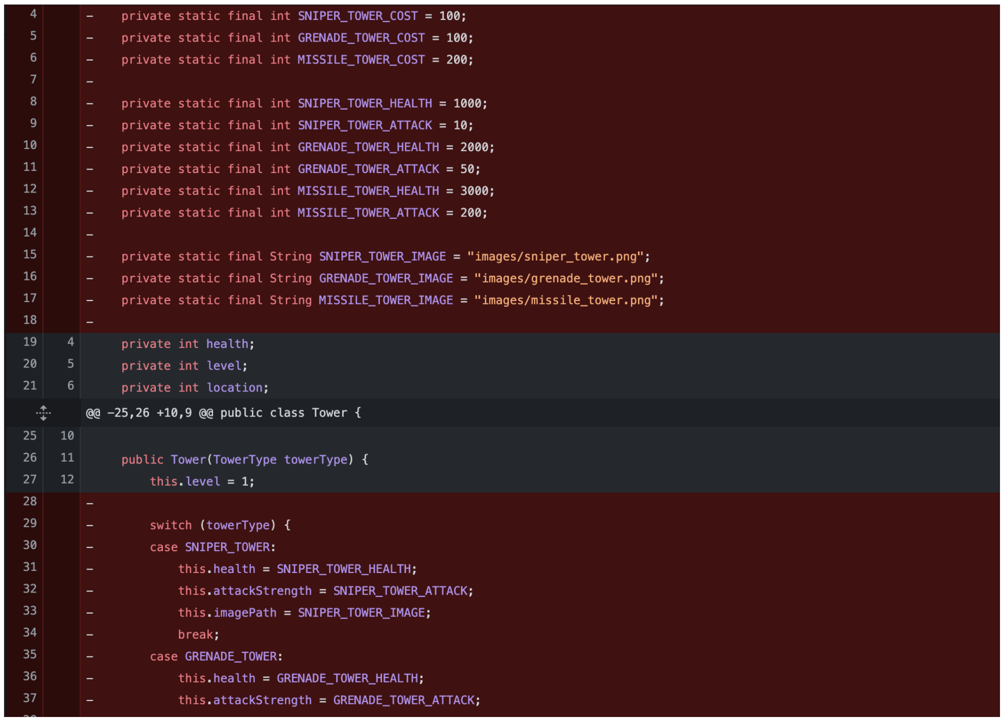
The getMoney() method in the Enemy class was deleted to fix this. The code won't be affected as a getHealth() method in the EnemyType class serves the same purpose.

Code Smell # 4 - Bloaters; Large Class

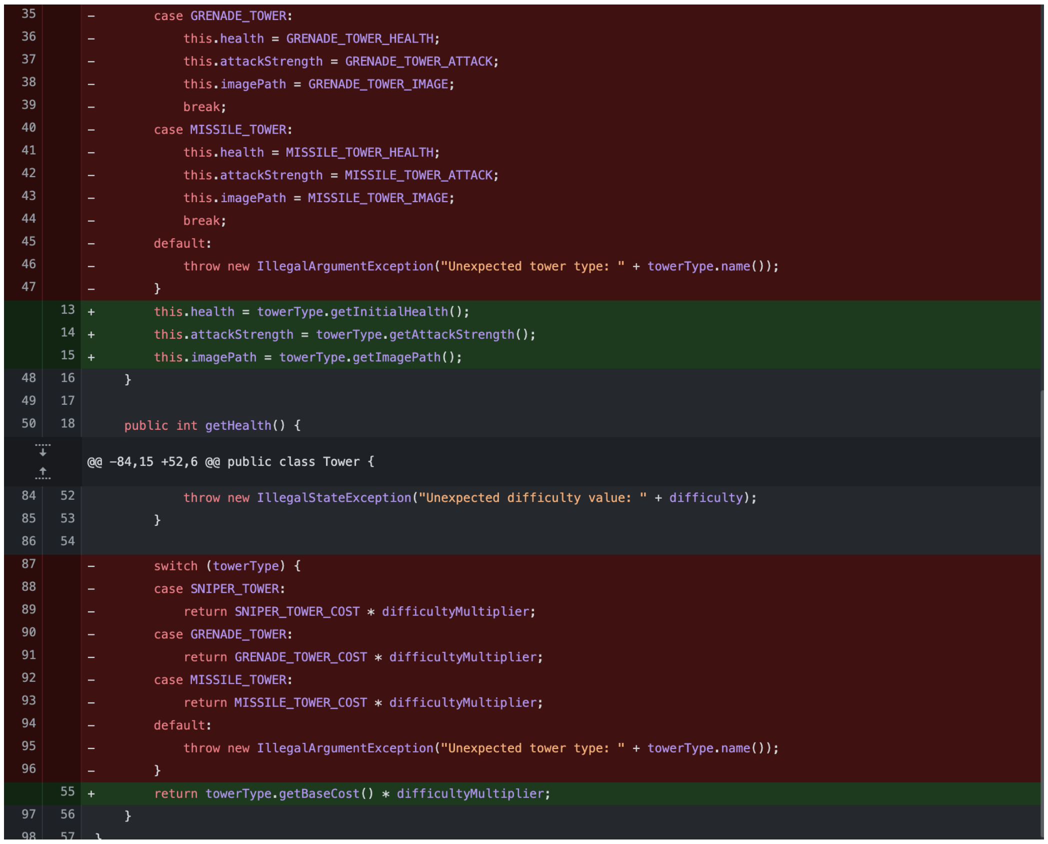
The Tower class contained a significant amount of information, including information about the towers and each type of tower. This class had an excessive amount of information and was deemed a large class. This code smell exists due to technical debt, as less time was spent putting all the code into one class.

The Tower class was divided into a TowerType Enum class and a Tower Class to fix this. The TowerType class contained information such as listing the different types of towers and the elements associated with that particular type of tower. Then the Tower class was used to instantiate the object and the image of each tower.

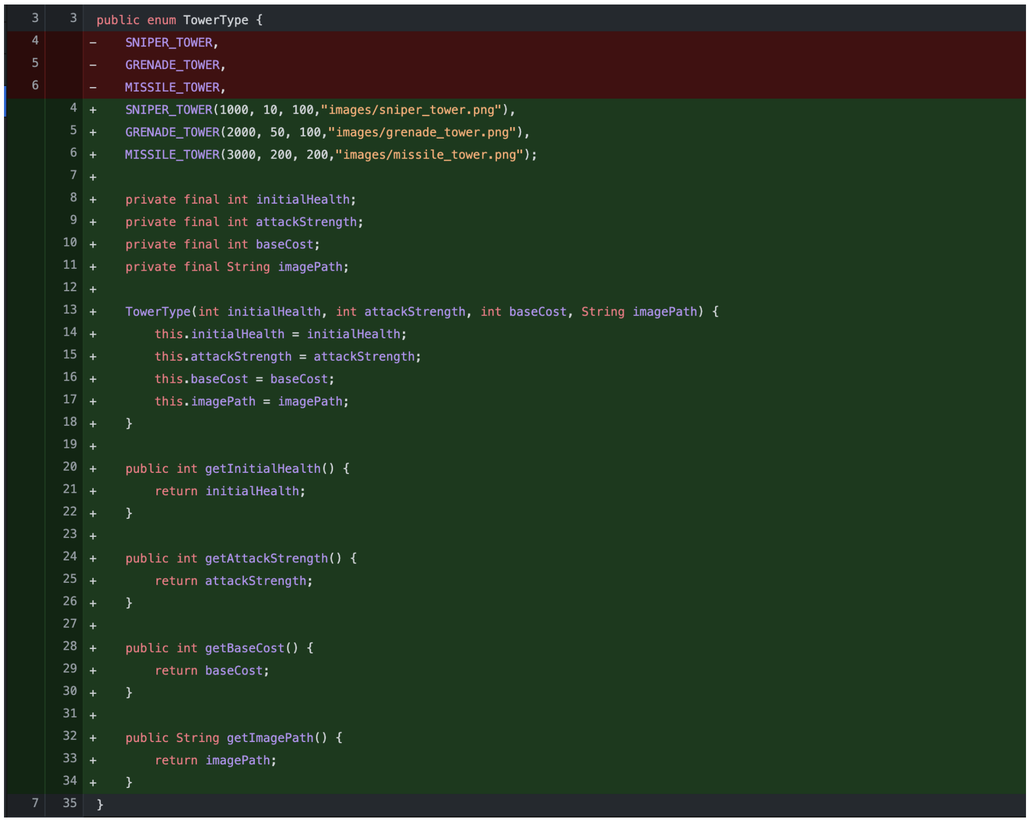
Seen below is the refactored code. The red sections are the sections of the code that were deleted and the green ections of the code are sections of the code that were added.



```diff
-    private static final int SNIPER_TOWER_COST = 100;
-    private static final int GRENADE_TOWER_COST = 100;
-    private static final int MISSILE_TOWER_COST = 200;
-
-    private static final int SNIPER_TOWER_HEALTH = 1000;
-    private static final int SNIPER_TOWER_ATTACK = 10;
-    private static final int GRENADE_TOWER_HEALTH = 2000;
-    private static final int GRENADE_TOWER_ATTACK = 50;
-    private static final int MISSILE_TOWER_HEALTH = 3000;
-    private static final int MISSILE_TOWER_ATTACK = 200;
-
-    private static final String SNIPER_TOWER_IMAGE = "images/sniper_tower.png";
-    private static final String GRENADE_TOWER_IMAGE = "images/grenade_tower.png";
-    private static final String MISSILE_TOWER_IMAGE = "images/missile_tower.png";
-
     private int health;
     private int level;
     private int location;
@@ -25,26 +10,9 @@ public class Tower {

     public Tower(TowerType towerType) {
         this.level = 1;
-
-        switch (towerType) {
-        case SNIPER_TOWER:
-            this.health = SNIPER_TOWER_HEALTH;
-            this.attackStrength = SNIPER_TOWER_ATTACK;
-            this.imagePath = SNIPER_TOWER_IMAGE;
-            break;
-        case GRENADE_TOWER:
-            this.health = GRENADE_TOWER_HEALTH;
-            this.attackStrength = GRENADE_TOWER_ATTACK;
```



```diff
-        case GRENADE_TOWER:
-            this.health = GRENADE_TOWER_HEALTH;
-            this.attackStrength = GRENADE_TOWER_ATTACK;
-            this.imagePath = GRENADE_TOWER_IMAGE;
-            break;
-        case MISSILE_TOWER:
-            this.health = MISSILE_TOWER_HEALTH;
-            this.attackStrength = MISSILE_TOWER_ATTACK;
-            this.imagePath = MISSILE_TOWER_IMAGE;
-            break;
-        default:
-            throw new IllegalArgumentException("Unexpected tower type: " + towerType.name());
-        }
+        this.health = towerType.getInitialHealth();
+        this.attackStrength = towerType.getAttackStrength();
+        this.imagePath = towerType.getImagePath();
     }

     public int getHealth() {
@@ -84,15 +52,6 @@ public class Tower {
             throw new IllegalStateException("Unexpected difficulty value: " + difficulty);
         }

-        switch (towerType) {
-        case SNIPER_TOWER:
-            return SNIPER_TOWER_COST * difficultyMultiplier;
-        case GRENADE_TOWER:
-            return GRENADE_TOWER_COST * difficultyMultiplier;
-        case MISSILE_TOWER:
-            return MISSILE_TOWER_COST * difficultyMultiplier;
-        default:
-            throw new IllegalArgumentException("Unexpected tower type: " + towerType.name());
-        }
+        return towerType.getBaseCost() * difficultyMultiplier;
     }
 }
```

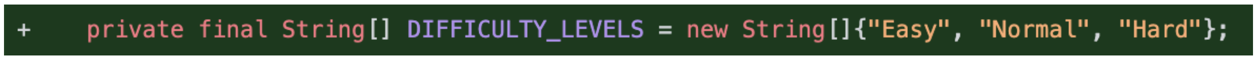


```diff
 public enum TowerType {
-    SNIPER_TOWER,
-    GRENADE_TOWER,
-    MISSILE_TOWER,
+    SNIPER_TOWER(1000, 10, 100,"images/sniper_tower.png"),
+    GRENADE_TOWER(2000, 50, 100,"images/grenade_tower.png"),
+    MISSILE_TOWER(3000, 200, 200,"images/missile_tower.png");
+
+    private final int initialHealth;
+    private final int attackStrength;
+    private final int baseCost;
+    private final String imagePath;
+
+    TowerType(int initialHealth, int attackStrength, int baseCost, String imagePath) {
+        this.initialHealth = initialHealth;
+        this.attackStrength = attackStrength;
+        this.baseCost = baseCost;
+        this.imagePath = imagePath;
+    }
+
+    public int getInitialHealth() {
+        return initialHealth;
+    }
+
+    public int getAttackStrength() {
+        return attackStrength;
+    }
+
+    public int getBaseCost() {
+        return baseCost;
+    }
+
+    public String getImagePath() {
+        return imagePath;
+    }
 }
```

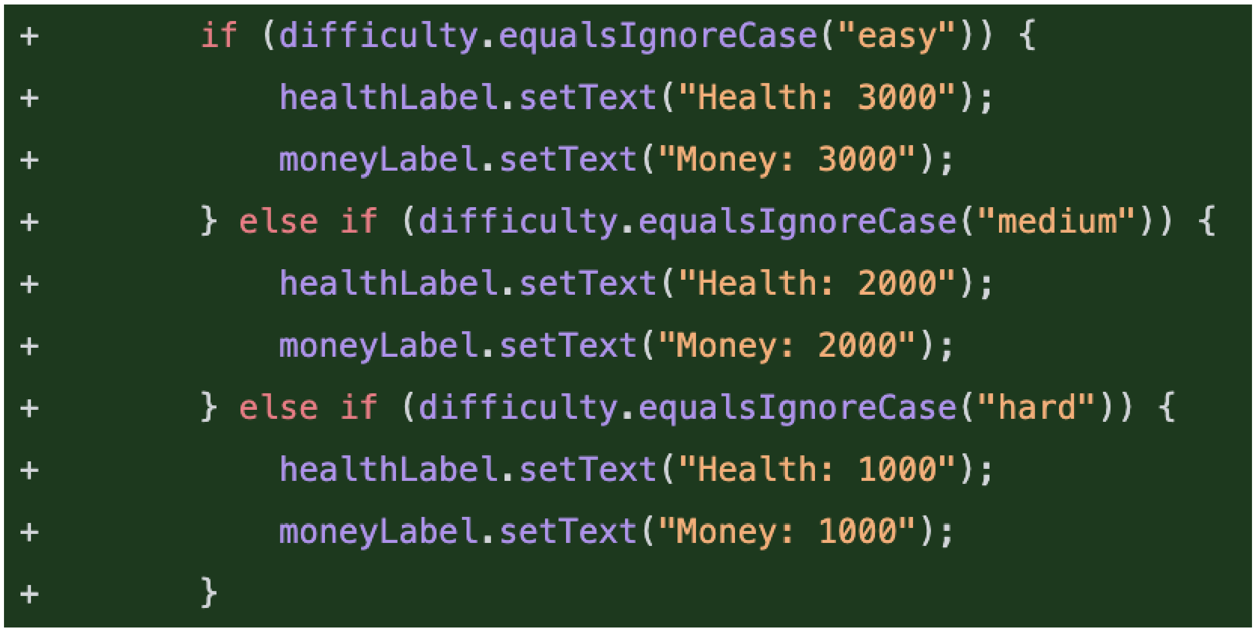
Code Smell # 5 - Bloaters; Primitive Obsession

Before we were using primitives like Strings to determine the difficulty level of the game. These String values were later repeated across different classes which caused a lot of redundancy but we needed it in multiple classes as it would determine how hard or easy it would be to purchase different towers and how much damage can be sustained.

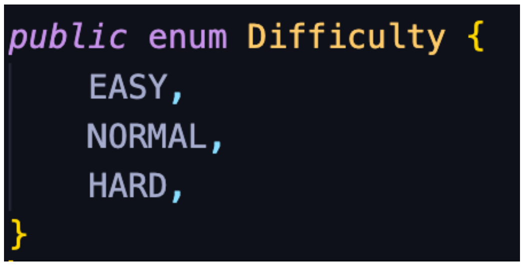
However, we realized it would be easier to have a single Enum class that will hold the different difficulties. With the Enum class, we no longer had to initialize the Strings by themselves and this resulted in a more comprehensive and easily readable code. It also provides more flexibility as we can potentially add more difficulties if we wanted to adapt more difficulties in the future which would make it easier to implement. Seen below is the example of how the code was prior to creating the Enum and how it was after creating the Enum.



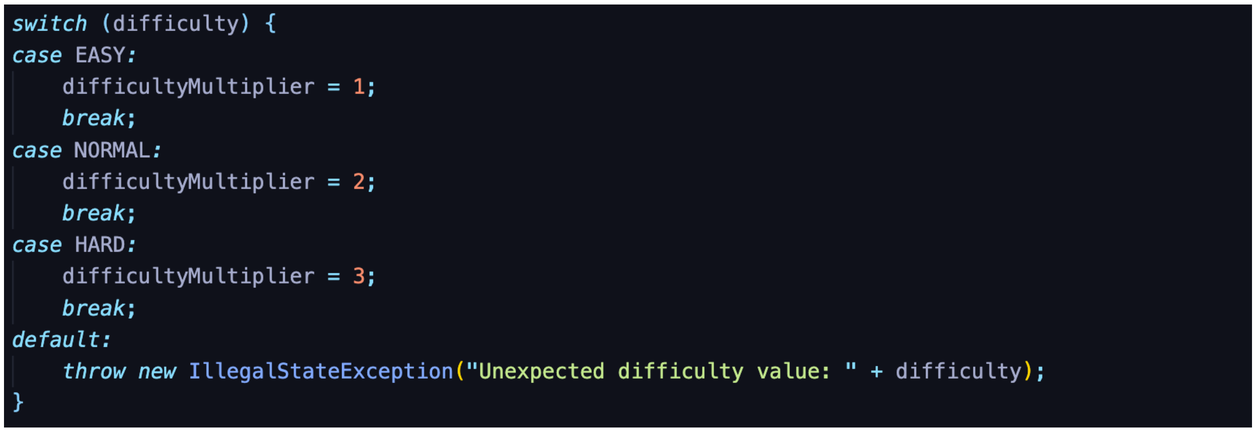
```
+    private final String[] DIFFICULTY_LEVELS = new String[]{"Easy", "Normal", "Hard"};
```



```
+         if (difficulty.equalsIgnoreCase("easy")) {
+             healthLabel.setText("Health: 3000");
+             moneyLabel.setText("Money: 3000");
+         } else if (difficulty.equalsIgnoreCase("medium")) {
+             healthLabel.setText("Health: 2000");
+             moneyLabel.setText("Money: 2000");
+         } else if (difficulty.equalsIgnoreCase("hard")) {
+             healthLabel.setText("Health: 1000");
+             moneyLabel.setText("Money: 1000");
+         }
```



```
public enum Difficulty {
    EASY,
    NORMAL,
    HARD,
}
```



```
switch (difficulty) {
case EASY:
    difficultyMultiplier = 1;
    break;
case NORMAL:
    difficultyMultiplier = 2;
    break;
case HARD:
    difficultyMultiplier = 3;
    break;
default:
    throw new IllegalStateException("Unexpected difficulty value: " + difficulty);
}
```