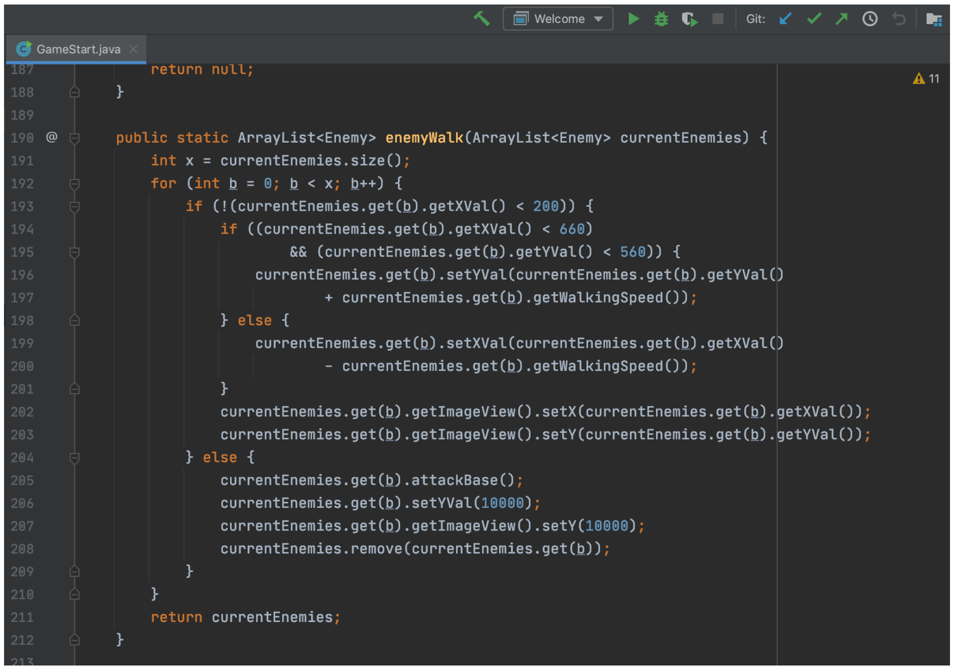
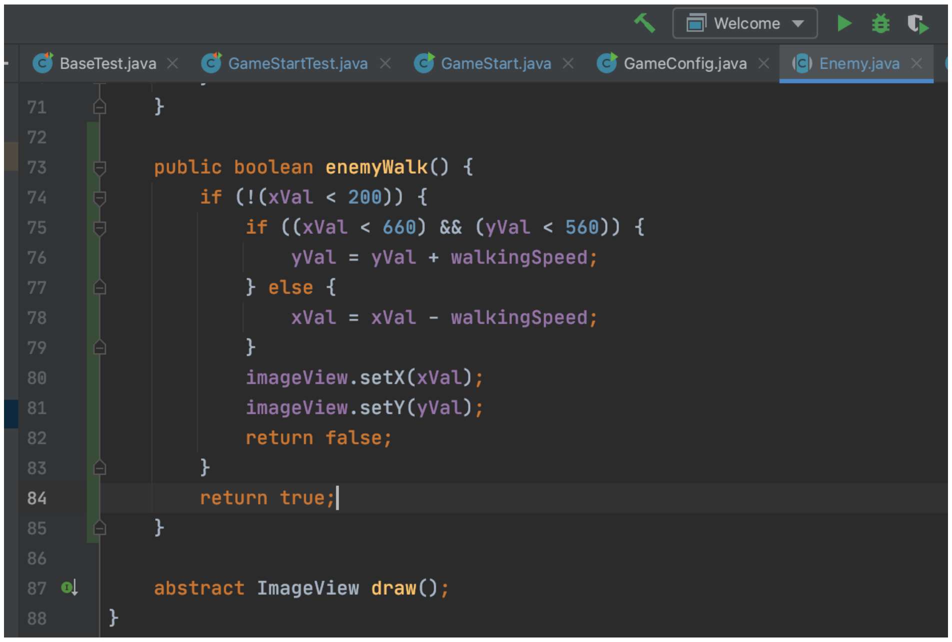
Feature Envy Code Smell - Erin



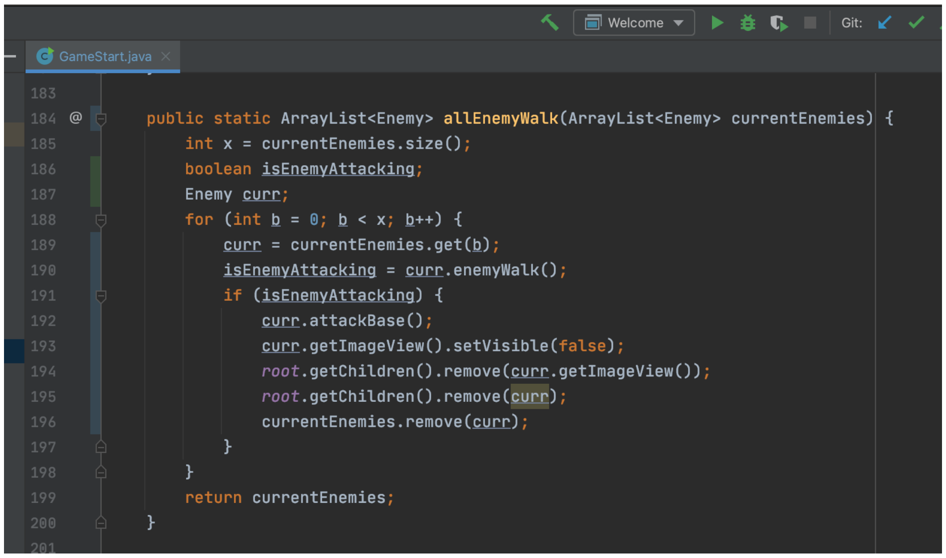
```java
        return null;
    }

    public static ArrayList<Enemy> enemyWalk(ArrayList<Enemy> currentEnemies) {
        int x = currentEnemies.size();
        for (int b = 0; b < x; b++) {
            if (!(currentEnemies.get(b).getXVal() < 200)) {
                if ((currentEnemies.get(b).getXVal() < 660)
                        && (currentEnemies.get(b).getYVal() < 560)) {
                    currentEnemies.get(b).setYVal(currentEnemies.get(b).getYVal()
                            + currentEnemies.get(b).getWalkingSpeed());
                } else {
                    currentEnemies.get(b).setXVal(currentEnemies.get(b).getXVal()
                            - currentEnemies.get(b).getWalkingSpeed());
                }
                currentEnemies.get(b).getImageView().setX(currentEnemies.get(b).getXVal());
                currentEnemies.get(b).getImageView().setY(currentEnemies.get(b).getYVal());
            } else {
                currentEnemies.get(b).attackBase();
                currentEnemies.get(b).setYVal(10000);
                currentEnemies.get(b).getImageView().setY(10000);
                currentEnemies.remove(currentEnemies.get(b));
            }
        }
        return currentEnemies;
    }
```

Fig. 1 - Feature Envy Code Smell in enemyWalk method
This method was originally in GameStart.java which contains the code for controlling the functionality of when the game starts to send enemies. The code's function is to change the location of the enemy based on where it is in the path. Because this code heavily uses variables located in the Enemy interface rather than the variables in the class it is contained in, this is an example of a feature envy code smell.



```java
    }

    public boolean enemyWalk() {
        if (!(xVal < 200)) {
            if ((xVal < 660) && (yVal < 560)) {
                yVal = yVal + walkingSpeed;
            } else {
                xVal = xVal - walkingSpeed;
            }
            imageView.setX(xVal);
            imageView.setY(yVal);
            return false;
        }
        return true;
    }

    abstract ImageView draw();
}
```

Figure 2. Moved enemyWalk() method to Enemy.java



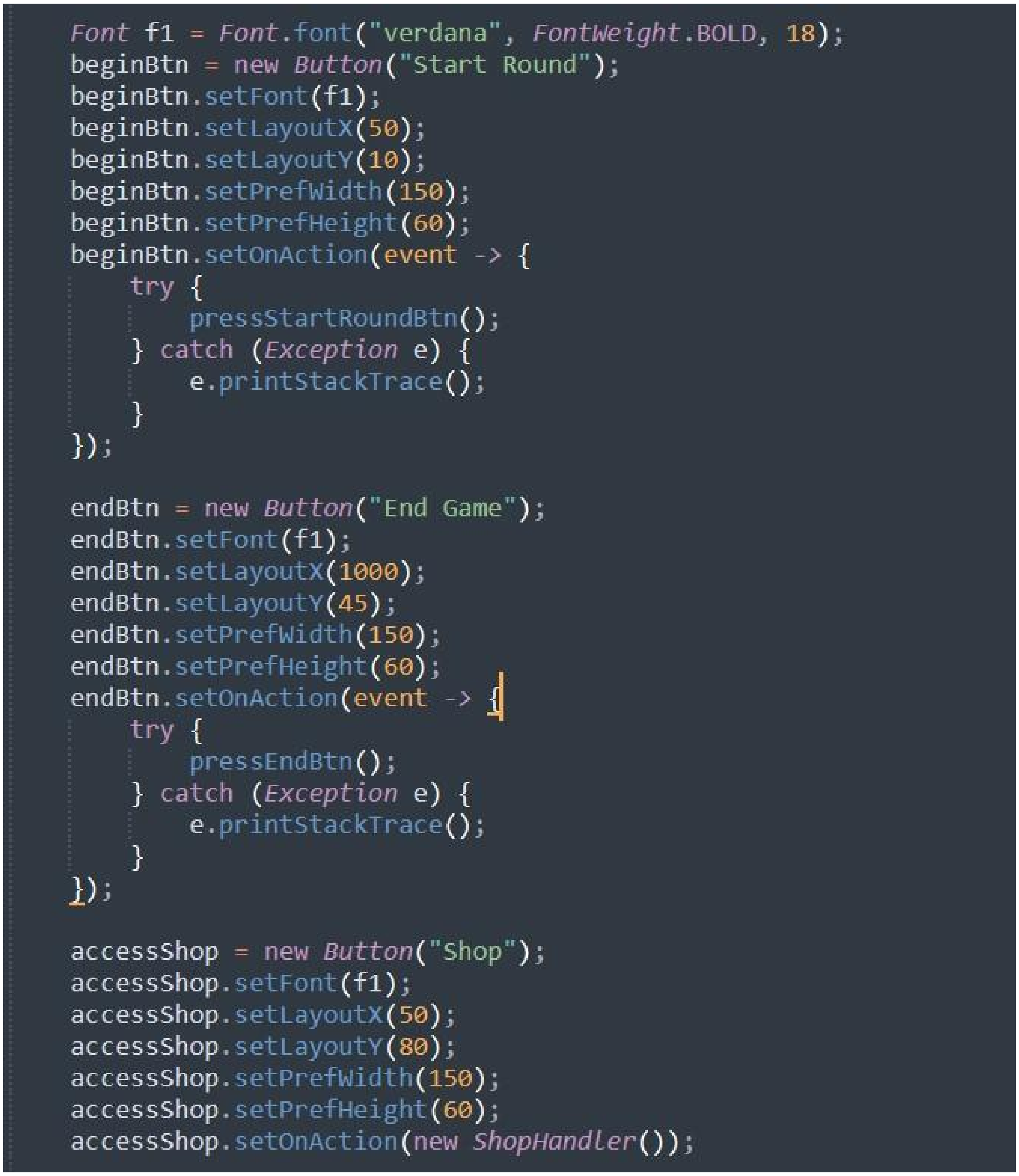
```java
    public static ArrayList<Enemy> allEnemyWalk(ArrayList<Enemy> currentEnemies) {
        int x = currentEnemies.size();
        boolean isEnemyAttacking;
        Enemy curr;
        for (int b = 0; b < x; b++) {
            curr = currentEnemies.get(b);
            isEnemyAttacking = curr.enemyWalk();
            if (isEnemyAttacking) {
                curr.attackBase();
                curr.getImageView().setVisible(false);
                root.getChildren().remove(curr.getImageView());
                root.getChildren().remove(curr);
                currentEnemies.remove(curr);
            }
        }
        return currentEnemies;
    }
```

Figure 3. Added allEnemyWalk to handle all current enemies walking

I fixed the feature envy code smell by changing the location of the enemyWalk method. I moved it from the GameStart.java to the Enemy.java Interface as seen in Figure 2 because most of the variables/methods called when the method was in the GameStart.java was from the Enemy.java interface. Therefore, it was more sensible to move the method to the Enemy.java interface. However, since there is an ArrayList of enemies in the GameStart.java, there was a need to create a new method that would call the enemyWalk method for each enemy in the currentEnemies. I had kept the lines of code that are currently in the Figure 3 if (isEnemyAttacking) statement because these lines needed elements that were in GameStart.java and needed to be kept together since they implement the enemy attacking and disappearing rather than walking.

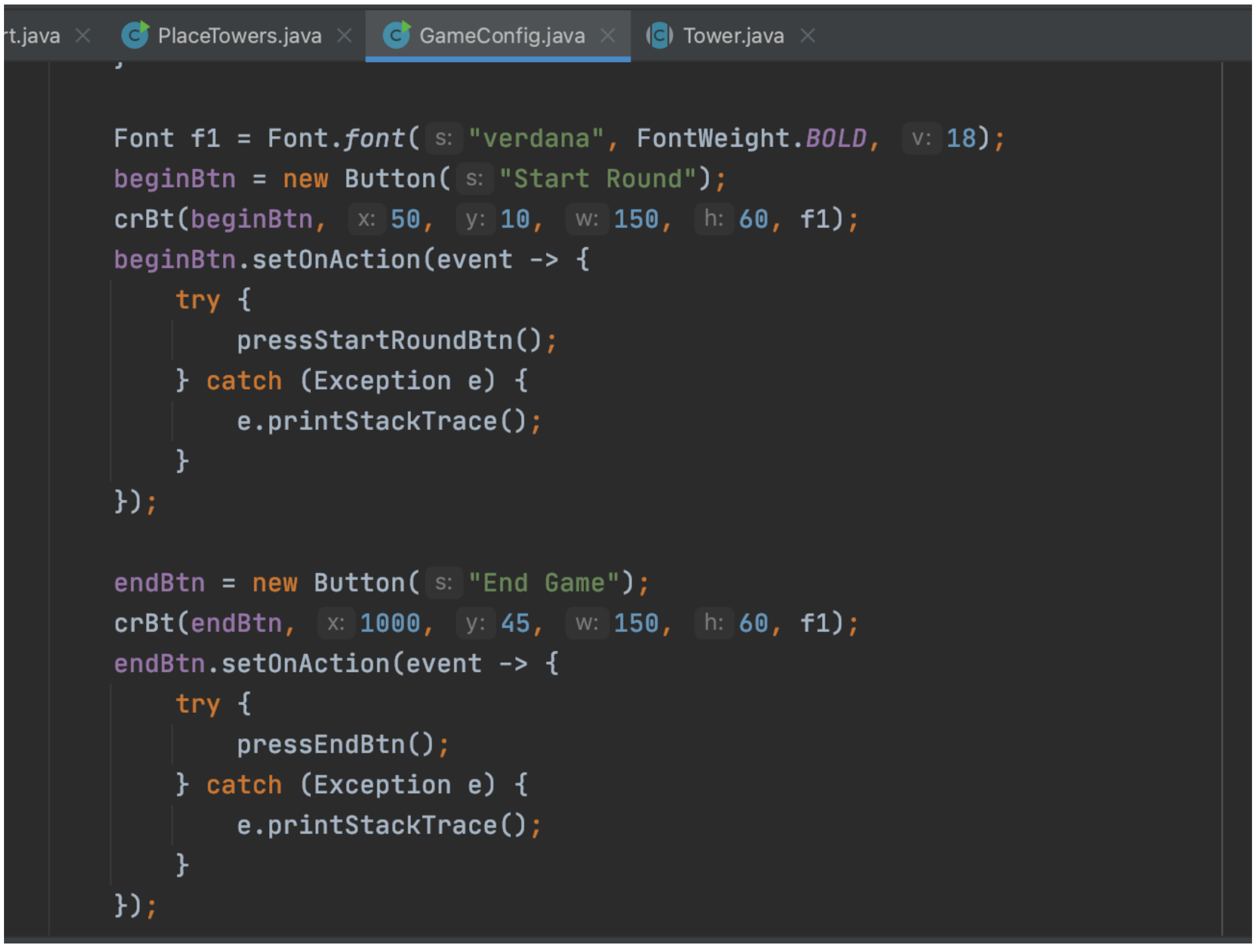
Duplicated Code Code Smell - Vishal

In most of the classes, many buttons are created on the screen. The process for creating these buttons is largely the same: Set the location and size of the button to their correct values:

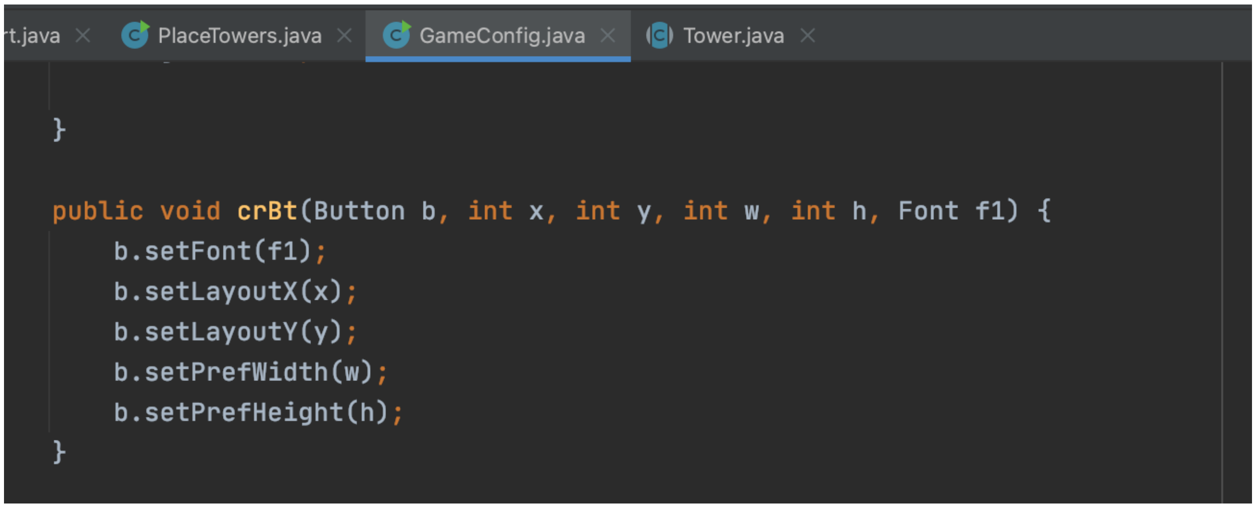


```java
Font f1 = Font.font("verdana", FontWeight.BOLD, 18);
beginBtn = new Button("Start Round");
beginBtn.setFont(f1);
beginBtn.setLayoutX(50);
beginBtn.setLayoutY(10);
beginBtn.setPrefWidth(150);
beginBtn.setPrefHeight(60);
beginBtn.setOnAction(event -> {
    try {
        pressStartRoundBtn();
    } catch (Exception e) {
        e.printStackTrace();
    }
});

endBtn = new Button("End Game");
endBtn.setFont(f1);
endBtn.setLayoutX(1000);
endBtn.setLayoutY(45);
endBtn.setPrefWidth(150);
endBtn.setPrefHeight(60);
endBtn.setOnAction(event -> {
    try {
        pressEndBtn();
    } catch (Exception e) {
        e.printStackTrace();
    }
});

accessShop = new Button("Shop");
accessShop.setFont(f1);
accessShop.setLayoutX(50);
accessShop.setLayoutY(80);
accessShop.setPrefWidth(150);
accessShop.setPrefHeight(60);
accessShop.setOnAction(new ShopHandler());
```

This is a duplicated code code smell, as we are duplicating the same button creation code every single time we want to create a button. This can be fixed by creating one method that sets the location and size of the button, and using it every time we want to create one:



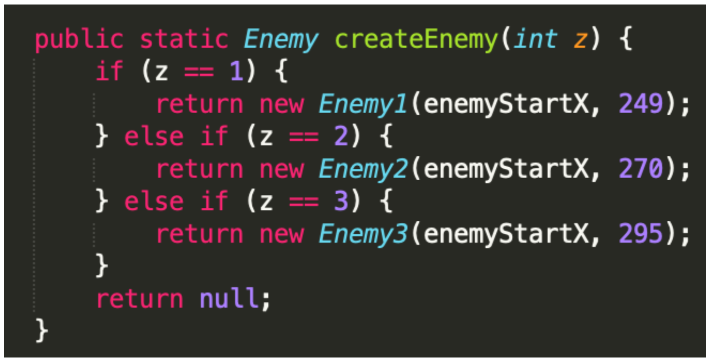
```java
    Font f1 = Font.font( s: "verdana", FontWeight.BOLD,  v: 18);
    beginBtn = new Button( s: "Start Round");
    crBt(beginBtn,  x: 50,  y: 10,  w: 150,  h: 60, f1);
    beginBtn.setOnAction(event -> {
        try {
            pressStartRoundBtn();
        } catch (Exception e) {
            e.printStackTrace();
        }
    });

    endBtn = new Button( s: "End Game");
    crBt(endBtn,  x: 1000,  y: 45,  w: 150,  h: 60, f1);
    endBtn.setOnAction(event -> {
        try {
            pressEndBtn();
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
```



```java
}

public void crBt(Button b, int x, int y, int w, int h, Font f1) {
    b.setFont(f1);
    b.setLayoutX(x);
    b.setLayoutY(y);
    b.setPrefWidth(w);
    b.setPrefHeight(h);
}
```

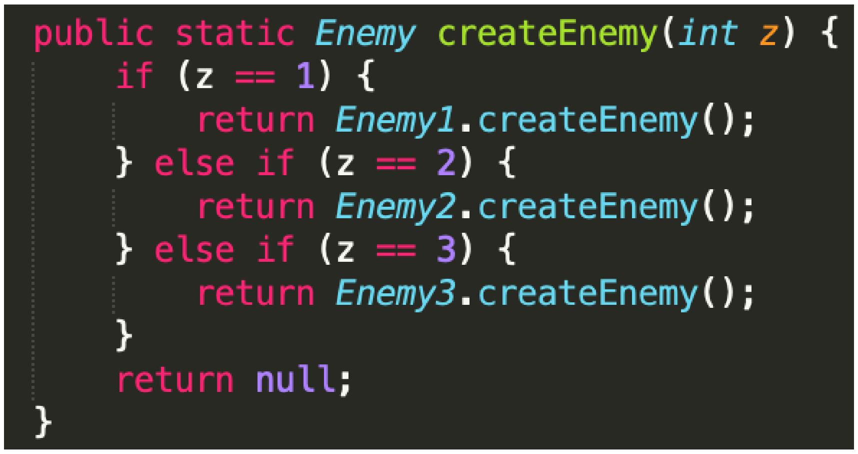
Object Oriented Abusers (switch-case/if statement code smell) - Anika

This if/else statement was originally in the GameStart class and is considered a code smell in OOP. This is because it violates the Open/Closed principle as adding a new condition would require modification to existing code and can be redundant and harder to keep track of as the method becomes longer from new conditions.

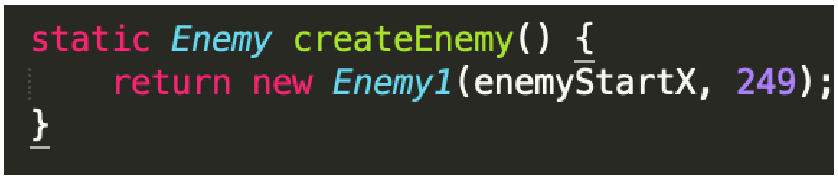


```
public static Enemy createEnemy(int z) {
    if (z == 1) {
        return new Enemy1(enemyStartX, 249);
    } else if (z == 2) {
        return new Enemy2(enemyStartX, 270);
    } else if (z == 3) {
        return new Enemy3(enemyStartX, 295);
    }
    return null;
}
```

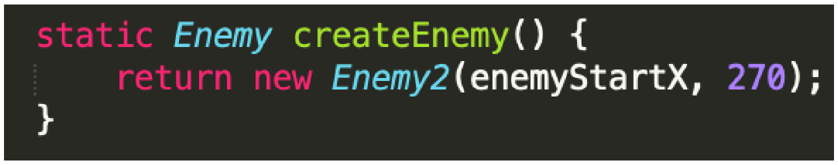
In order to align with OOP principles and use polymorphism, we can use if/else statements to create objects instead. Each object is created in its corresponding subclass rather than in one main class.



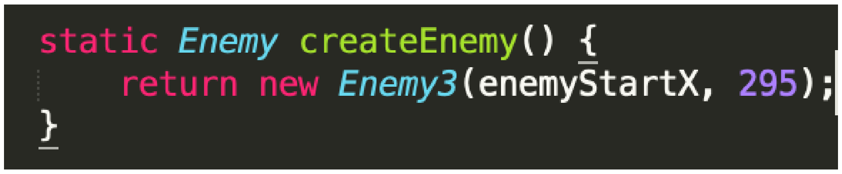
```
public static Enemy createEnemy(int z) {
    if (z == 1) {
        return Enemy1.createEnemy();
    } else if (z == 2) {
        return Enemy2.createEnemy();
    } else if (z == 3) {
        return Enemy3.createEnemy();
    }
    return null;
}
```



```
static Enemy createEnemy() {
    return new Enemy1(enemyStartX, 249);
}
```



```
static Enemy createEnemy() {
    return new Enemy2(enemyStartX, 270);
}
```



```
static Enemy createEnemy() {
    return new Enemy3(enemyStartX, 295);
}
```

Poorly Named Identifiers code smells - Tuan

```
public double distCalculator (Tower t,Enemy e) {
       if (t == null || e == null) {
       return -1;
       }
       // else calculate the distance between the two objects
       double t1 = t.getXVal();
       double t2= t.getYVal();
       double e1 = e.getXVal();
       double e2 = e.getYVal();

       double y = Math.abs(e2 - t2);
       double z = Math.abs(e1 - t1);

       return Math.hypot(y, z);
}
```

- This function is used to calculate the distance between the tower and the enemy. The function itself deals with a lot of variables which are coordinates of the two objects. The names of those variables for those x, y values were so hard to understand, people reading this code won't have any clue what the code is doing because the name of variables is meaningless.

- To fix this, the variable names are changed to the coordinate value and which object is holding it. The readers now can understand this code is trying to do a Pythagorean theorem or distance formula calculation from inputs that belong to each object.

```
public double distCalculator(Enemy e) {
   if (e == null) {
       return -1.0;
   }
```

```java
    double towerX = xVal;
    double towerY = yVal;
    double enemyX = e.getXVal();
    double enemyY = e.getYVal();

    double changeInY = Math.abs(enemyY - towerY);
    double changeInX = Math.abs(enemyX - towerX);
    // distance formula: sqrt(x^2 + y^2)
    return Math.hypot(changeInY, changeInX);
}
```