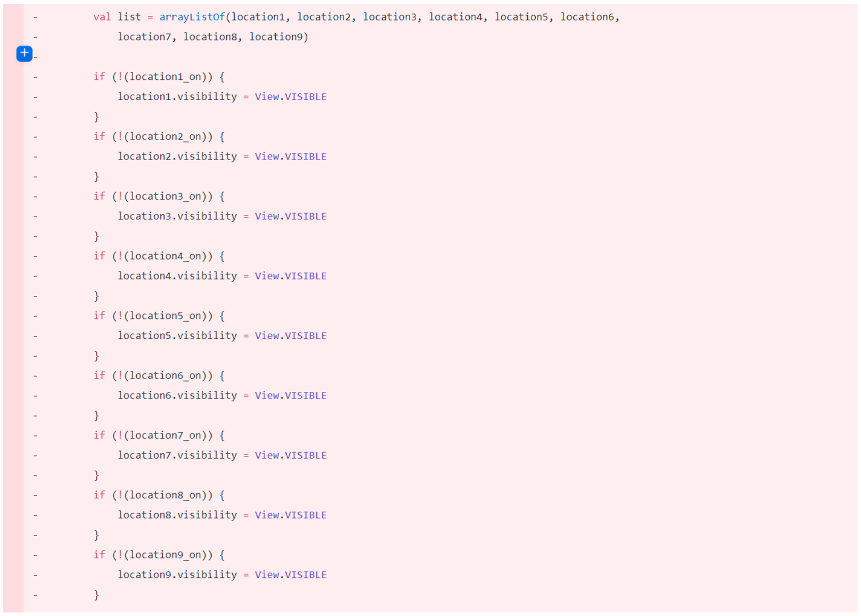
Large Class

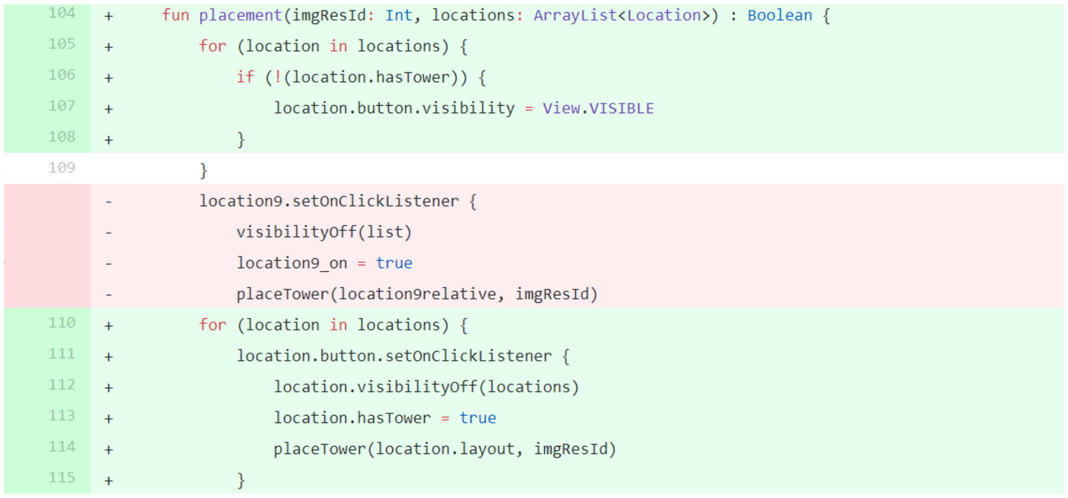
For this, we had essentially had copy-paste code for each tower placement location. Obviously, this made our class large, but it also made the game screen class do too much unrelated functionality.



```diff
-        val list = arrayListOf(location1, location2, location3, location4, location5, location6,
-            location7, location8, location9)
-
-        if (!(location1_on)) {
-            location1.visibility = View.VISIBLE
-        }
-        if (!(location2_on)) {
-            location2.visibility = View.VISIBLE
-        }
-        if (!(location3_on)) {
-            location3.visibility = View.VISIBLE
-        }
-        if (!(location4_on)) {
-            location4.visibility = View.VISIBLE
-        }
-        if (!(location5_on)) {
-            location5.visibility = View.VISIBLE
-        }
-        if (!(location6_on)) {
-            location6.visibility = View.VISIBLE
-        }
-        if (!(location7_on)) {
-            location7.visibility = View.VISIBLE
-        }
-        if (!(location8_on)) {
-            location8.visibility = View.VISIBLE
-        }
-        if (!(location9_on)) {
-            location9.visibility = View.VISIBLE
-        }
```

Fix:

The fix was to create a method that looped through all the placement locations (that was passed in as an array) and do the required functionality inside the method. Additionally, some things were moved to other classes, such as setting all of the placements to invisible was moved to the Map class.

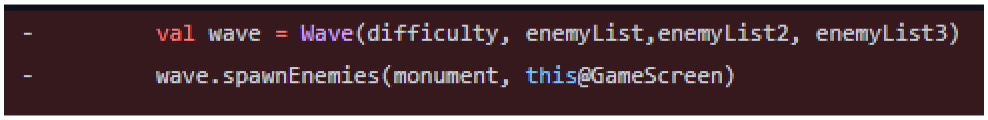


```diff
+    fun placement(imgResId: Int, locations: ArrayList<Location>) : Boolean {
+        for (location in locations) {
+            if (!(location.hasTower)) {
+                location.button.visibility = View.VISIBLE
+            }
         }
-        location9.setOnClickListener {
-            visibilityOff(list)
-            location9_on = true
-            placeTower(location9relative, imgResId)
+        for (location in locations) {
+            location.button.setOnClickListener {
+                location.visibilityOff(locations)
+                location.hasTower = true
+                placeTower(location.layout, imgResId)
+            }
```

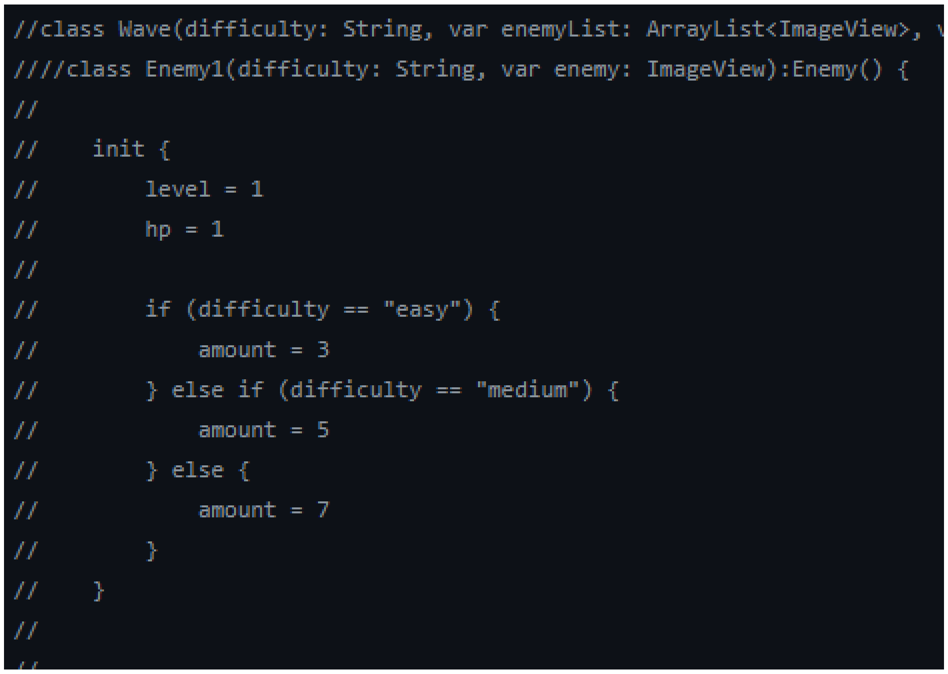
Divergent Change
It is difficult to screenshot exactly where divergent change occurred, so we will be using an explanation. Initially the GameScreen class was responsible for many things, meaning that whenever something was not working, the GameScreen class had to be changed. This resulted in many lines of code within that one class being edited.

Fix:
This code smell was alleviated by creating more classes to both distribute responsibilities accordingly and represent needed objects of the overall structure of the game. This reduced the number of changes that needed to be made to the GameScreen class itself.

Lazy Class



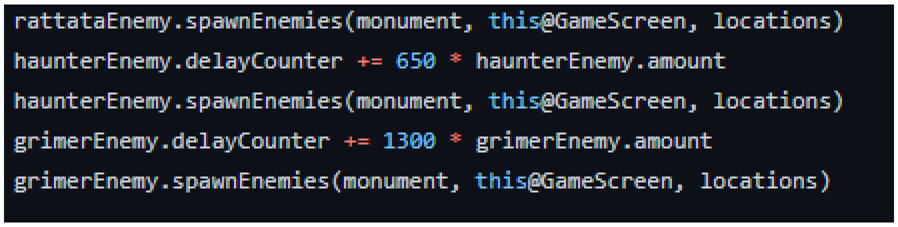
```
-         val wave = Wave(difficulty, enemyList,enemyList2, enemyList3)
-         wave.spawnEnemies(monument, this@GameScreen)
```



```
//class Wave(difficulty: String, var enemyList: ArrayList<ImageView>, v
////class Enemy1(difficulty: String, var enemy: ImageView):Enemy() {
//
//    init {
//        level = 1
//        hp = 1
//
//        if (difficulty == "easy") {
//            amount = 3
//        } else if (difficulty == "medium") {
//            amount = 5
//        } else {
//            amount = 7
//        }
//    }
//
```

The wave class appeared as a way to quickly determine if animating enemies was doable, as a proof of concept. Scalability was not taken into consideration, and the class was only needed to spawn enemies (nothing else).

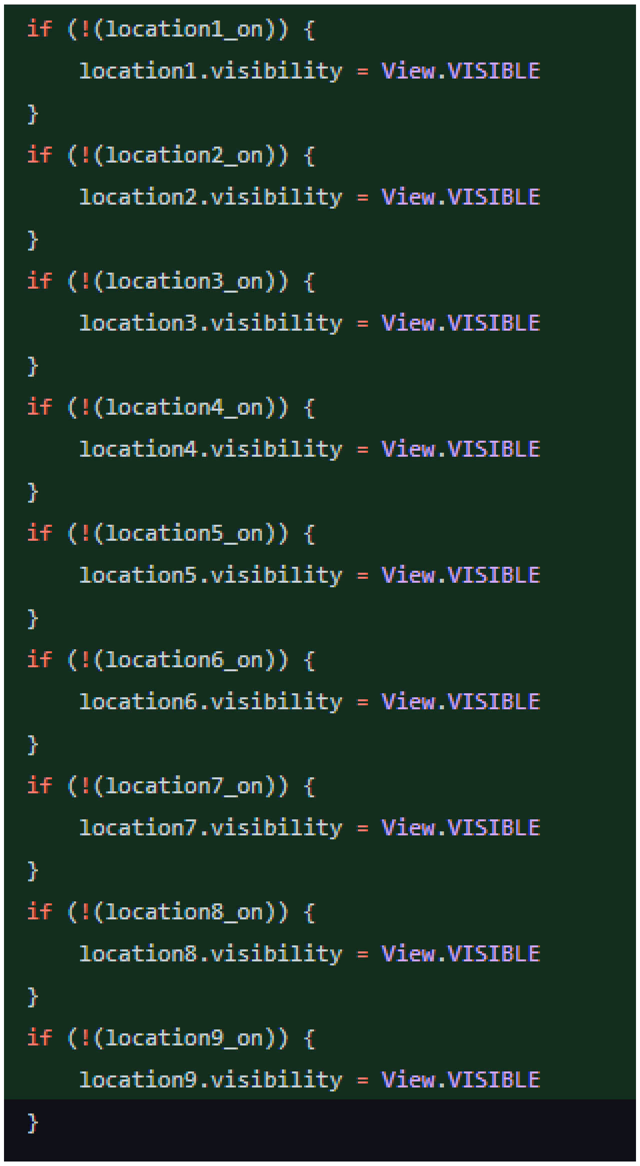
Fix:



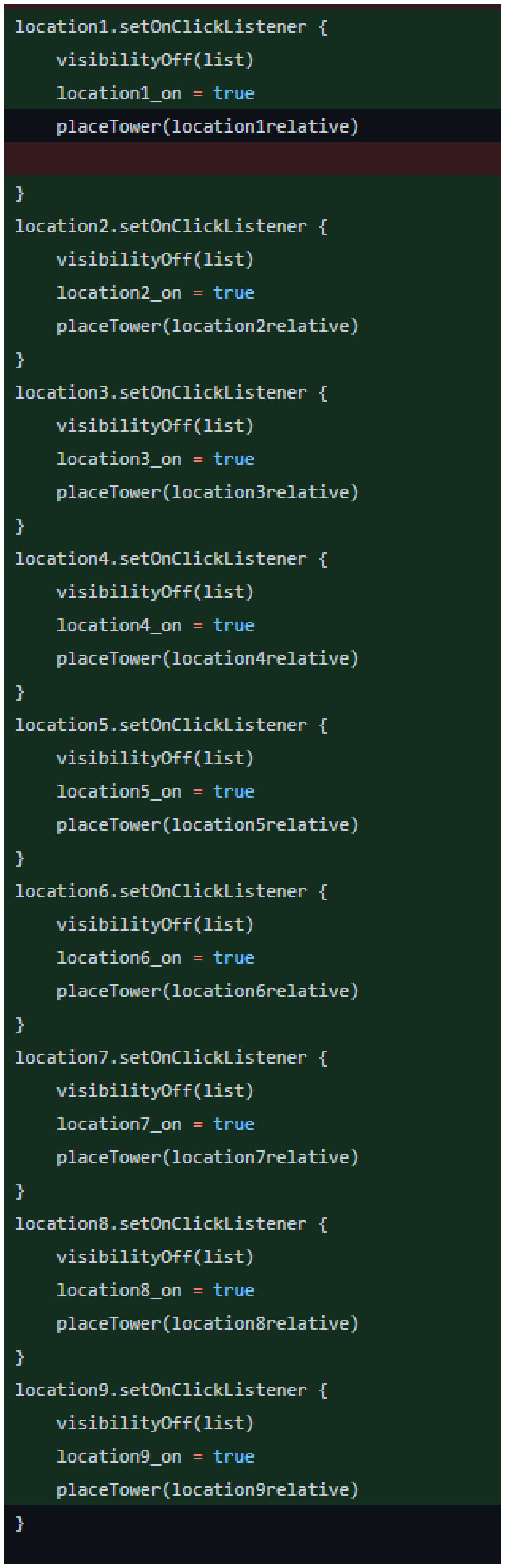
```
rattataEnemy.spawnEnemies(monument, this@GameScreen, locations)
haunterEnemy.delayCounter += 650 * haunterEnemy.amount
haunterEnemy.spawnEnemies(monument, this@GameScreen, locations)
grimerEnemy.delayCounter += 1300 * grimerEnemy.amount
grimerEnemy.spawnEnemies(monument, this@GameScreen, locations)
```

Although it is more lines of code, an unnecessary class was deleted and replaced with the spawning of each individual enemy object. These enemy objects are necessary for more than just spawning, meaning that the wave class was meaningless and useless.

Duplicated Code



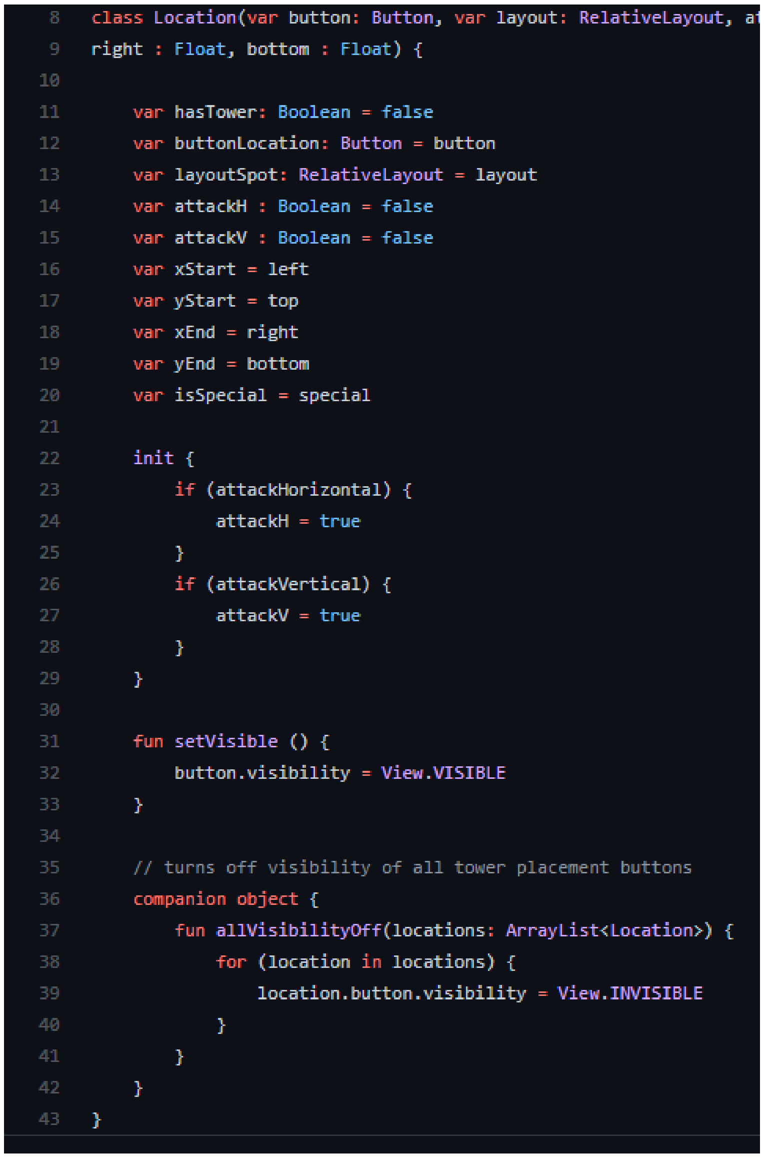
```
if (!(location1_on)) {
    location1.visibility = View.VISIBLE
}
if (!(location2_on)) {
    location2.visibility = View.VISIBLE
}
if (!(location3_on)) {
    location3.visibility = View.VISIBLE
}
if (!(location4_on)) {
    location4.visibility = View.VISIBLE
}
if (!(location5_on)) {
    location5.visibility = View.VISIBLE
}
if (!(location6_on)) {
    location6.visibility = View.VISIBLE
}
if (!(location7_on)) {
    location7.visibility = View.VISIBLE
}
if (!(location8_on)) {
    location8.visibility = View.VISIBLE
}
if (!(location9_on)) {
    location9.visibility = View.VISIBLE
}
```



```
location1.setOnClickListener {
    visibilityOff(list)
    location1_on = true
    placeTower(location1relative)

}
location2.setOnClickListener {
    visibilityOff(list)
    location2_on = true
    placeTower(location2relative)
}
location3.setOnClickListener {
    visibilityOff(list)
    location3_on = true
    placeTower(location3relative)
}
location4.setOnClickListener {
    visibilityOff(list)
    location4_on = true
    placeTower(location4relative)
}
location5.setOnClickListener {
    visibilityOff(list)
    location5_on = true
    placeTower(location5relative)
}
location6.setOnClickListener {
    visibilityOff(list)
    location6_on = true
    placeTower(location6relative)
}
location7.setOnClickListener {
    visibilityOff(list)
    location7_on = true
    placeTower(location7relative)
}
location8.setOnClickListener {
    visibilityOff(list)
    location8_on = true
    placeTower(location8relative)
}
location9.setOnClickListener {
    visibilityOff(list)
    location9_on = true
    placeTower(location9relative)
}
```

This code smell exists because our group was eager to implement the placing of a tower. Thus, to quickly create something working, this code was created. The idea was used as a proof of concept rather than the long term solution.

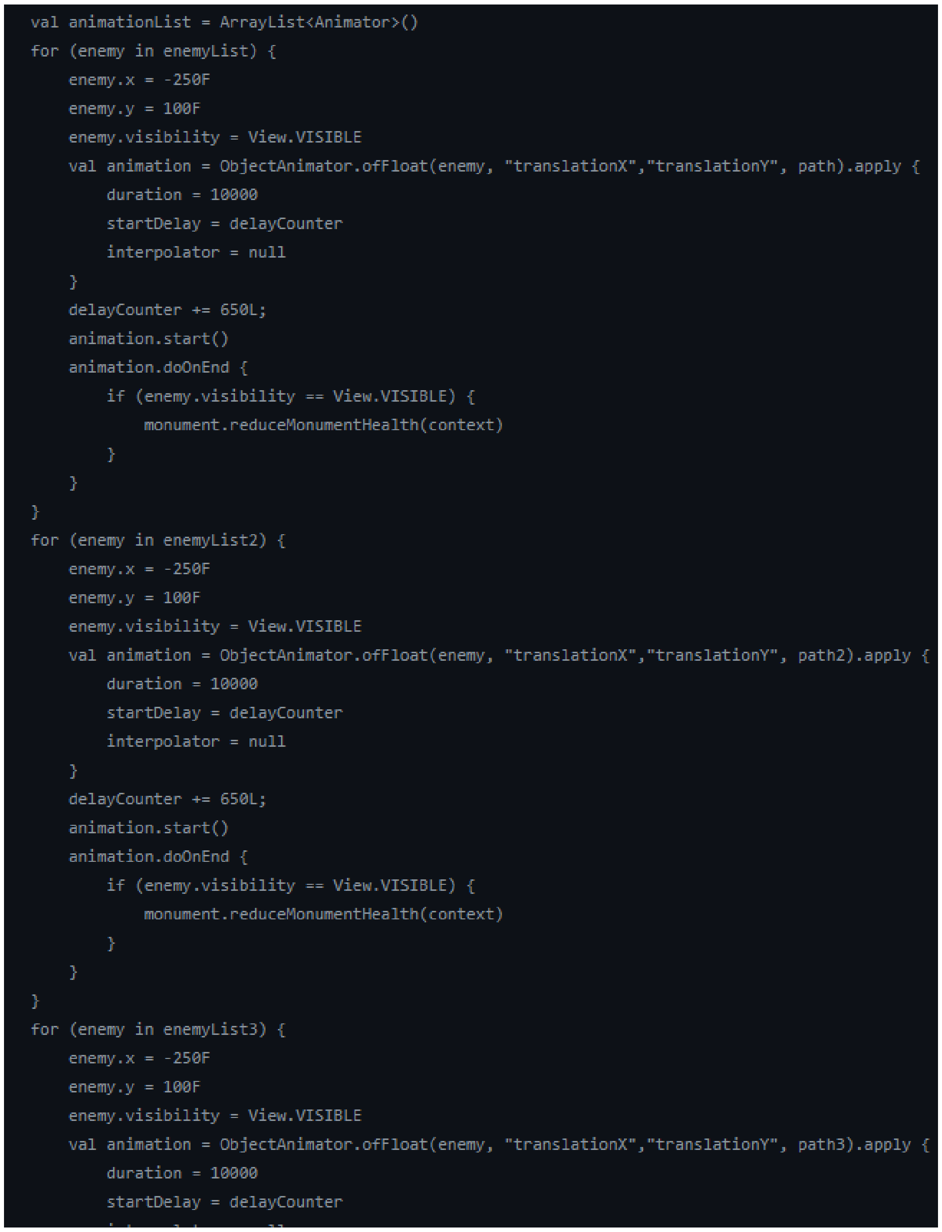
Fix:



```kotlin
class Location(var button: Button, var layout: RelativeLayout, a
right : Float, bottom : Float) {

    var hasTower: Boolean = false
    var buttonLocation: Button = button
    var layoutSpot: RelativeLayout = layout
    var attackH : Boolean = false
    var attackV : Boolean = false
    var xStart = left
    var yStart = top
    var xEnd = right
    var yEnd = bottom
    var isSpecial = special

    init {
        if (attackHorizontal) {
            attackH = true
        }
        if (attackVertical) {
            attackV = true
        }
    }

    fun setVisible () {
        button.visibility = View.VISIBLE
    }

    // turns off visibility of all tower placement buttons
    companion object {
        fun allVisibilityOff(locations: ArrayList<Location>) {
            for (location in locations) {
                location.button.visibility = View.INVISIBLE
            }
        }
    }
}
```

The code smell was fixed by making location an object and transferring the logic into the location class. This allowed us to remove the duplicated code with a place to set the visibility of each location rather than manually doing it to all of them line by line.

Feature envy



```
val animationList = ArrayList<Animator>()
for (enemy in enemyList) {
    enemy.x = -250F
    enemy.y = 100F
    enemy.visibility = View.VISIBLE
    val animation = ObjectAnimator.ofFloat(enemy, "translationX","translationY", path).apply {
        duration = 10000
        startDelay = delayCounter
        interpolator = null
    }
    delayCounter += 650L;
    animation.start()
    animation.doOnEnd {
        if (enemy.visibility == View.VISIBLE) {
            monument.reduceMonumentHealth(context)
        }
    }
}
for (enemy in enemyList2) {
    enemy.x = -250F
    enemy.y = 100F
    enemy.visibility = View.VISIBLE
    val animation = ObjectAnimator.ofFloat(enemy, "translationX","translationY", path2).apply {
        duration = 10000
        startDelay = delayCounter
        interpolator = null
    }
    delayCounter += 650L;
    animation.start()
    animation.doOnEnd {
        if (enemy.visibility == View.VISIBLE) {
            monument.reduceMonumentHealth(context)
        }
    }
}
for (enemy in enemyList3) {
    enemy.x = -250F
    enemy.y = 100F
    enemy.visibility = View.VISIBLE
    val animation = ObjectAnimator.ofFloat(enemy, "translationX","translationY", path3).apply {
        duration = 10000
        startDelay = delayCounter
```

The wave class only dealt with the enemy objects and nothing to do with any sort of "wave" object. This code smell originated to hastily satisfy the requirements instead of take long term considerations of the entire project into account.

Fix:
Through refactoring, the wave class was deleted. Instead, each enemy was made into a class and was represented as its own object, solving the problem of having a class that only dealt with enemies while not being an enemy itself.