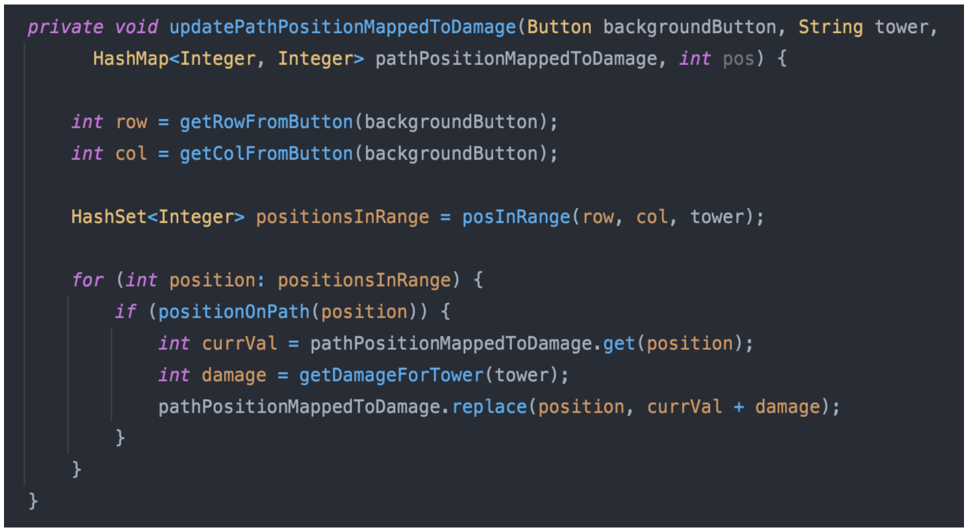
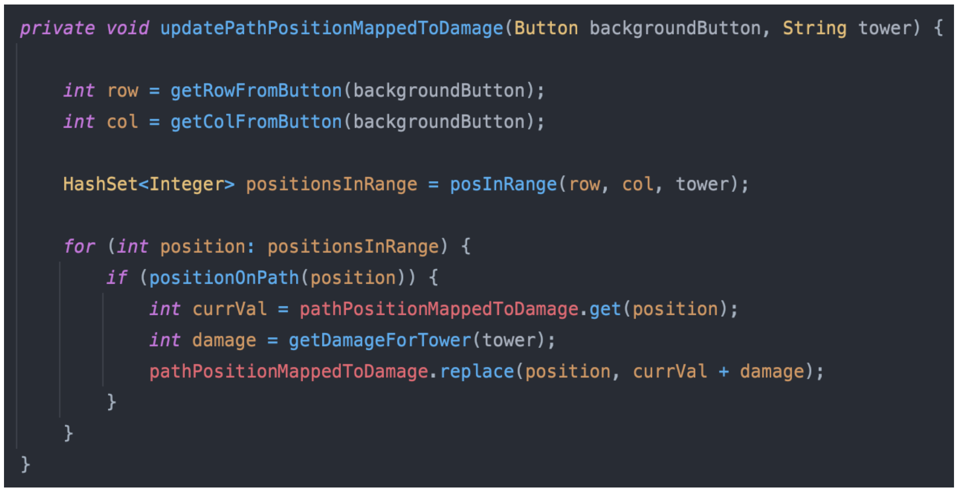
Code Smells Write-up

Code Smell 1: Bloaters → Long Parameter List



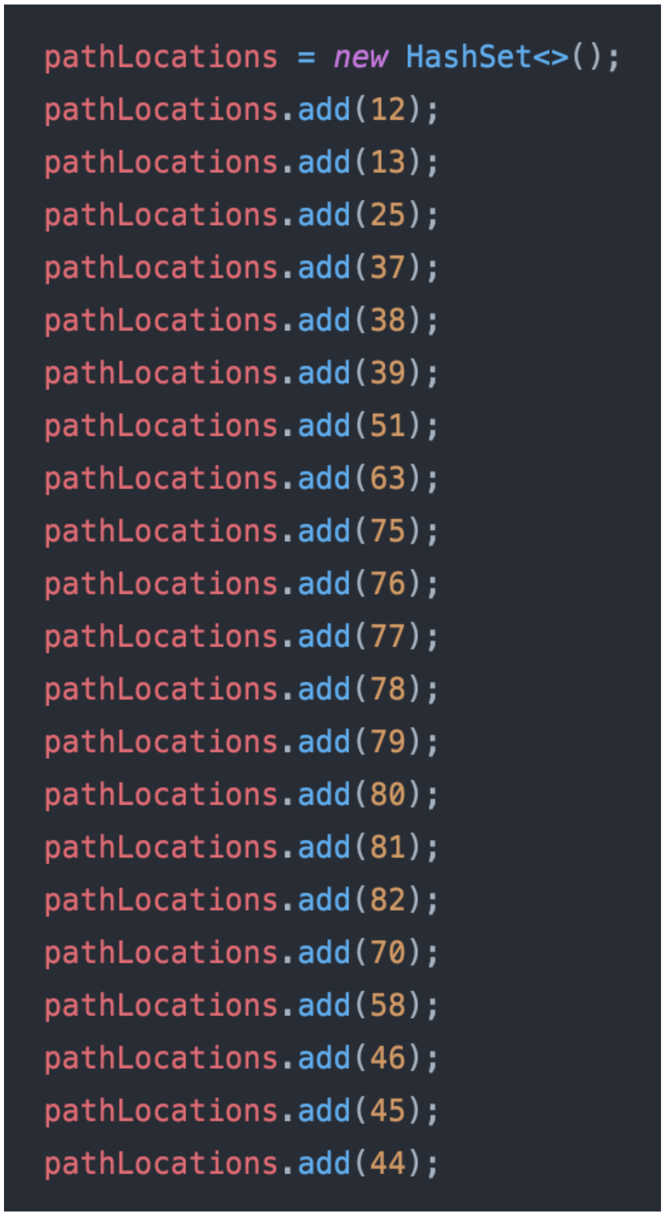
```java
private void updatePathPositionMappedToDamage(Button backgroundButton, String tower,
        HashMap<Integer, Integer> pathPositionMappedToDamage, int pos) {

    int row = getRowFromButton(backgroundButton);
    int col = getColFromButton(backgroundButton);

    HashSet<Integer> positionsInRange = posInRange(row, col, tower);

    for (int position: positionsInRange) {
        if (positionOnPath(position)) {
            int currVal = pathPositionMappedToDamage.get(position);
            int damage = getDamageForTower(tower);
            pathPositionMappedToDamage.replace(position, currVal + damage);
        }
    }
}
```



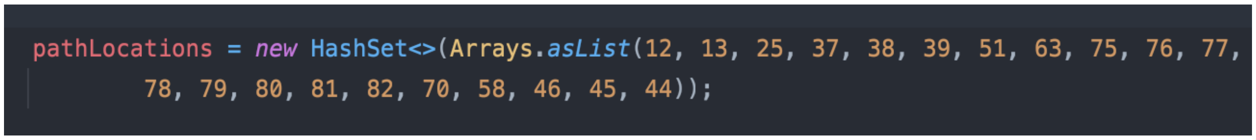
```java
private void updatePathPositionMappedToDamage(Button backgroundButton, String tower) {

    int row = getRowFromButton(backgroundButton);
    int col = getColFromButton(backgroundButton);

    HashSet<Integer> positionsInRange = posInRange(row, col, tower);

    for (int position: positionsInRange) {
        if (positionOnPath(position)) {
            int currVal = pathPositionMappedToDamage.get(position);
            int damage = getDamageForTower(tower);
            pathPositionMappedToDamage.replace(position, currVal + damage);
        }
    }
}
```

We changed our method in the above code and had redundant parameters that were being passed in. We were also able to edit the code to get rid of an additional parameter that we were using before, to cut down the total parameters used in this method from 4 to 2.

Code Smell 2: Dispensables → Duplicated Code



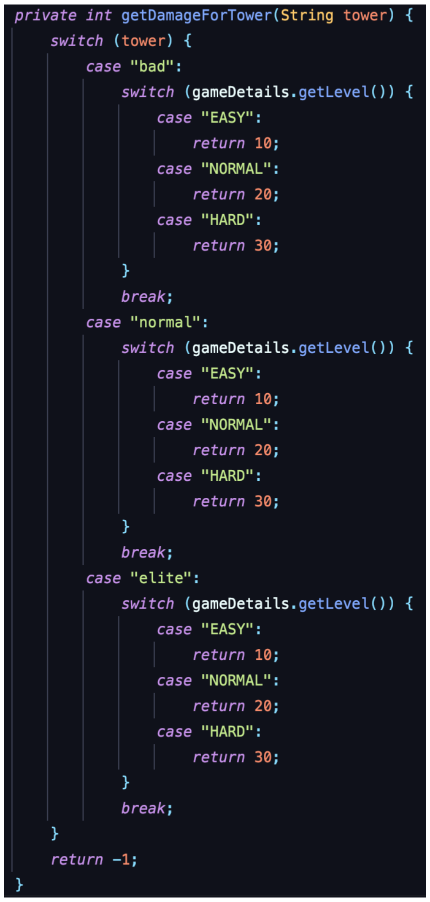
```java
pathLocations = new HashSet<>();
pathLocations.add(12);
pathLocations.add(13);
pathLocations.add(25);
pathLocations.add(37);
pathLocations.add(38);
pathLocations.add(39);
pathLocations.add(51);
pathLocations.add(63);
pathLocations.add(75);
pathLocations.add(76);
pathLocations.add(77);
pathLocations.add(78);
pathLocations.add(79);
pathLocations.add(80);
pathLocations.add(81);
pathLocations.add(82);
pathLocations.add(70);
pathLocations.add(58);
pathLocations.add(46);
pathLocations.add(45);
pathLocations.add(44);
```



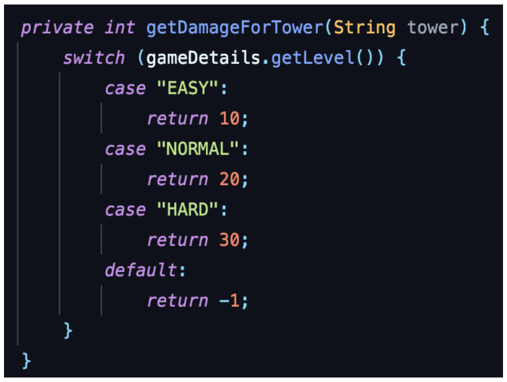
```java
pathLocations = new HashSet<>(Arrays.asList(12, 13, 25, 37, 38, 39, 51, 63, 75, 76, 77,
        78, 79, 80, 81, 82, 70, 58, 46, 45, 44));
```

We added a bunch of locations to pathLocations line by line, which led to it taking up a lot of space in the codebase and also led to duplicated code. We passed in the locations we wanted as a collection to pathLocations when initializing it so that we could get rid of the duplicated code.

Code Smell 3: Object-Orientation Abusers → Switch Statements



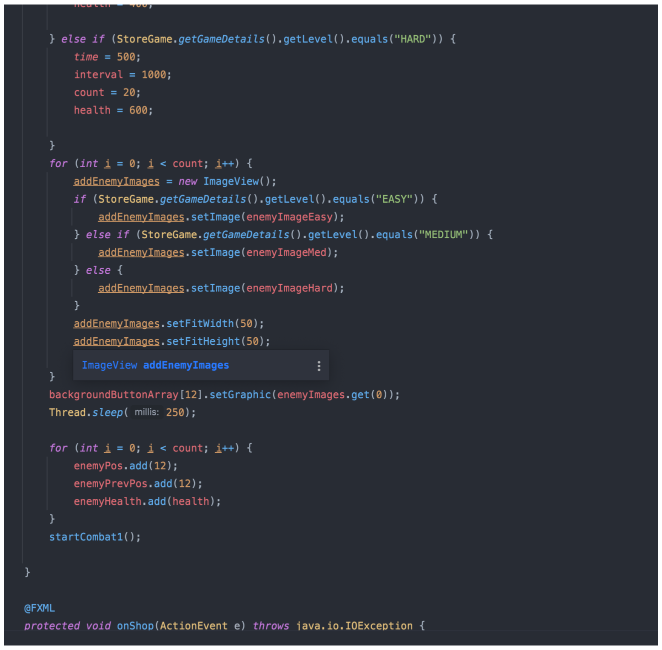
```java
private int getDamageForTower(String tower) {
    switch (tower) {
        case "bad":
            switch (gameDetails.getLevel()) {
                case "EASY":
                    return 10;
                case "NORMAL":
                    return 20;
                case "HARD":
                    return 30;
            }
            break;
        case "normal":
            switch (gameDetails.getLevel()) {
                case "EASY":
                    return 10;
                case "NORMAL":
                    return 20;
                case "HARD":
                    return 30;
            }
            break;
        case "elite":
            switch (gameDetails.getLevel()) {
                case "EASY":
                    return 10;
                case "NORMAL":
                    return 20;
                case "HARD":
                    return 30;
            }
            break;
    }
    return -1;
}
```



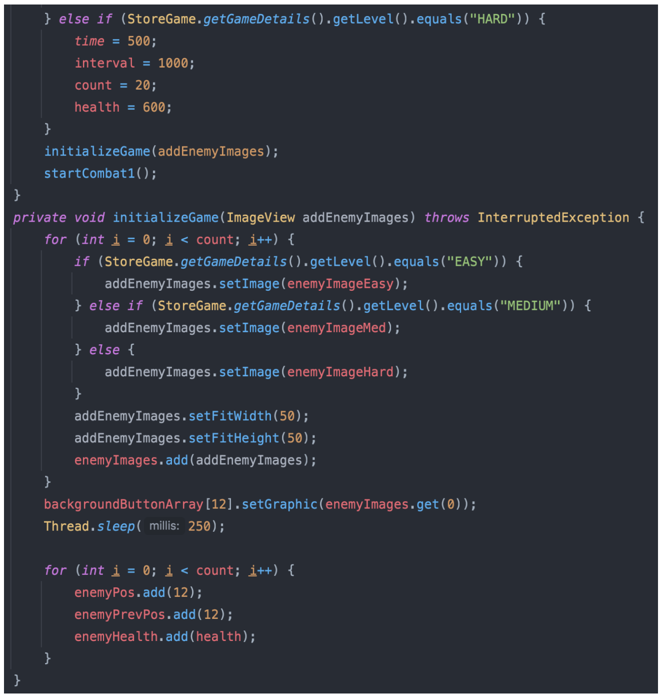
```java
private int getDamageForTower(String tower) {
    switch (gameDetails.getLevel()) {
        case "EASY":
            return 10;
        case "NORMAL":
            return 20;
        case "HARD":
            return 30;
        default:
            return -1;
    }
}
```

We realized that switching on the tower level was not required because our return value only depended on the level of difficulty of the game after we made changes to our game logic. Upon this realization, we changed our method to switch only on the level of the game, which is far more concise, clean, and uses better logic. Not only was our original code abusing object oriented programming constructs, but it was also duplicating code.

Code Smell 4: Bloaters → Long Method



```
        health = 400;

    } else if (StoreGame.getGameDetails().getLevel().equals("HARD")) {
        time = 500;
        interval = 1000;
        count = 20;
        health = 600;

    }
    for (int i = 0; i < count; i++) {
        addEnemyImages = new ImageView();
        if (StoreGame.getGameDetails().getLevel().equals("EASY")) {
            addEnemyImages.setImage(enemyImageEasy);
        } else if (StoreGame.getGameDetails().getLevel().equals("MEDIUM")) {
            addEnemyImages.setImage(enemyImageMed);
        } else {
            addEnemyImages.setImage(enemyImageHard);
        }
        addEnemyImages.setFitWidth(50);
        addEnemyImages.setFitHeight(50);
        ImageView addEnemyImages
    }
    backgroundButtonArray[12].setGraphic(enemyImages.get(0));
    Thread.sleep( millis: 250);

    for (int i = 0; i < count; i++) {
        enemyPos.add(12);
        enemyPrevPos.add(12);
        enemyHealth.add(health);
    }
    startCombat1();

}

@FXML
protected void onShop(ActionEvent e) throws java.io.IOException {
```

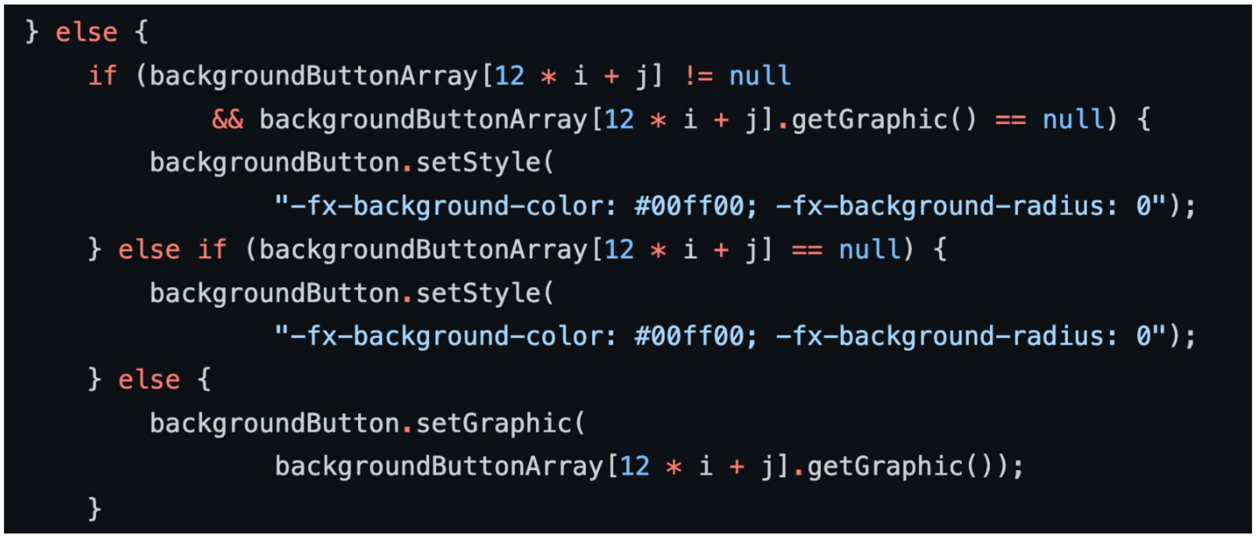


```
    } else if (StoreGame.getGameDetails().getLevel().equals("HARD")) {
        time = 500;
        interval = 1000;
        count = 20;
        health = 600;
    }
    initializeGame(addEnemyImages);
    startCombat1();
}
private void initializeGame(ImageView addEnemyImages) throws InterruptedException {
    for (int i = 0; i < count; i++) {
        if (StoreGame.getGameDetails().getLevel().equals("EASY")) {
            addEnemyImages.setImage(enemyImageEasy);
        } else if (StoreGame.getGameDetails().getLevel().equals("MEDIUM")) {
            addEnemyImages.setImage(enemyImageMed);
        } else {
            addEnemyImages.setImage(enemyImageHard);
        }
        addEnemyImages.setFitWidth(50);
        addEnemyImages.setFitHeight(50);
        enemyImages.add(addEnemyImages);
    }
    backgroundButtonArray[12].setGraphic(enemyImages.get(0));
    Thread.sleep( millis: 250);

    for (int i = 0; i < count; i++) {
        enemyPos.add(12);
        enemyPrevPos.add(12);
        enemyHealth.add(health);
    }
}
```

Our “onPlay()” method that executes when the user clicks the play button was long and performed too many different operations on the game and its state. We wanted to make this method especially readable since it is part of our core functionality. Thus, we moved a portion of the code into a new method named initializeGame(). This new method initializes the game with the appropriate parameters and presets before we begin combat.

Code Smell 5: Change Preventers → Shotgun Surgery

During M3, when we were trying to place our towers onto the map, we kept running into issues with our game turning a weird color and freezing because we were using button.setDisable(false) and button.setDisable(true), which caused weird behavior in the game. We didn't realize at first that this was the issue, so we kept making a bunch of small modifications to a few different files since we had too much coupling: namely Landscape Controller, ShopController, and GameDetails.



```
} else {
    if (backgroundButtonArray[12 * i + j] != null
            && backgroundButtonArray[12 * i + j].getGraphic() == null) {
        backgroundButton.setStyle(
                "-fx-background-color: #00ff00; -fx-background-radius: 0");
    } else if (backgroundButtonArray[12 * i + j] == null) {
        backgroundButton.setStyle(
                "-fx-background-color: #00ff00; -fx-background-radius: 0");
    } else {
        backgroundButton.setGraphic(
                backgroundButtonArray[12 * i + j].getGraphic());
    }
```

It got annoying chasing around these small errors, so instead we went a different route and decided not to use setDisable(). We ran a for loop over the grid and instead just checked which spots already had a tower by checking if there was a graphic present. This code fixed the issue with the freezing and the changing color and also allowed us to restore our old code.