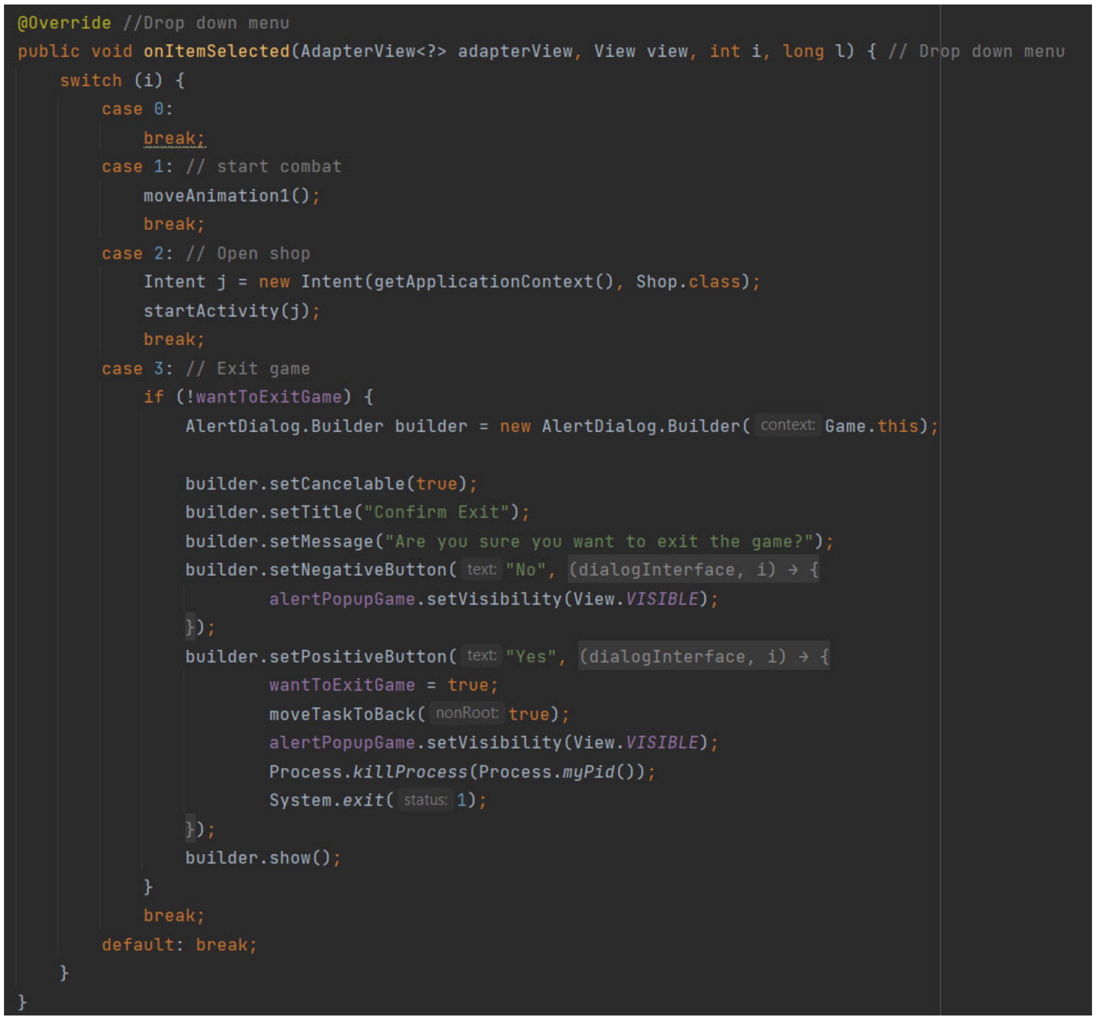
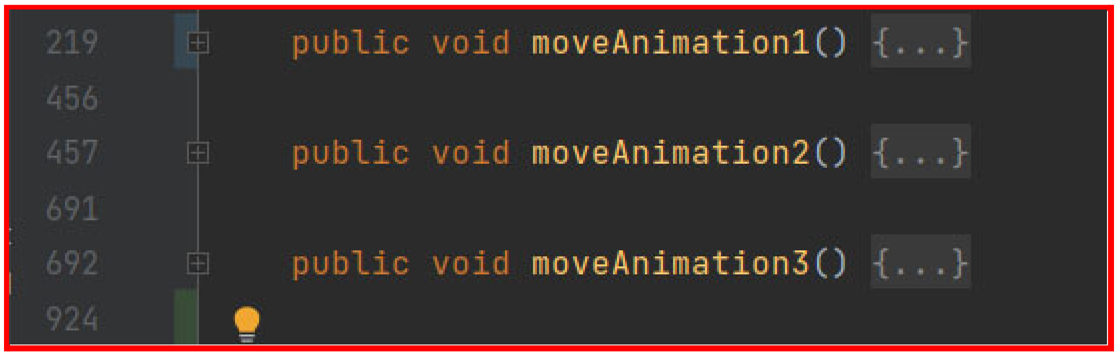
1) Object-Orientation Abusers: Switch Statements



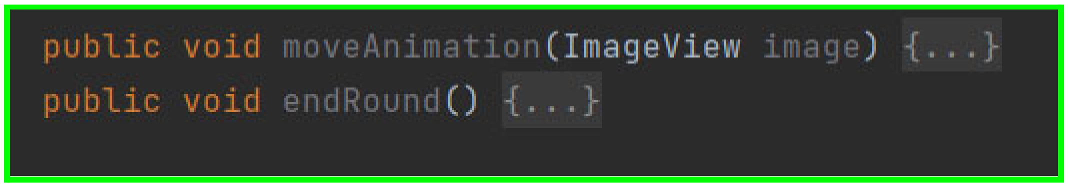
```
@Override //Drop down menu
public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) { // Drop down menu
    switch (i) {
        case 0:
            break;
        case 1: // start combat
            moveAnimation1();
            break;
        case 2: // Open shop
            Intent j = new Intent(getApplicationContext(), Shop.class);
            startActivity(j);
            break;
        case 3: // Exit game
            if (!wantToExitGame) {
                AlertDialog.Builder builder = new AlertDialog.Builder( context: Game.this);

                builder.setCancelable(true);
                builder.setTitle("Confirm Exit");
                builder.setMessage("Are you sure you want to exit the game?");
                builder.setNegativeButton( text: "No", (dialogInterface, i) -> {
                        alertPopupGame.setVisibility(View.VISIBLE);
                });
                builder.setPositiveButton( text: "Yes", (dialogInterface, i) -> {
                        wantToExitGame = true;
                        moveTaskToBack( nonRoot: true);
                        alertPopupGame.setVisibility(View.VISIBLE);
                        Process.killProcess(Process.myPid());
                        System.exit( status: 1);
                });
                builder.show();
            }
            break;
        default: break;
    }
}
```

In order to switch between the menu options in the in-game menu, a switch statement was used to differentiate the options, the first being a break, allowing the player to exit out of the menu, the second being the start combat option, the next button opens the shop, and the last one brings up a menu allowing the user to exit the game. This is necessary to the code because it allows the users to navigate the menu.

2) Bloaters: Long Method



```java
public void moveAnimation1() {...}

public void moveAnimation2() {...}

public void moveAnimation3() {...}

```

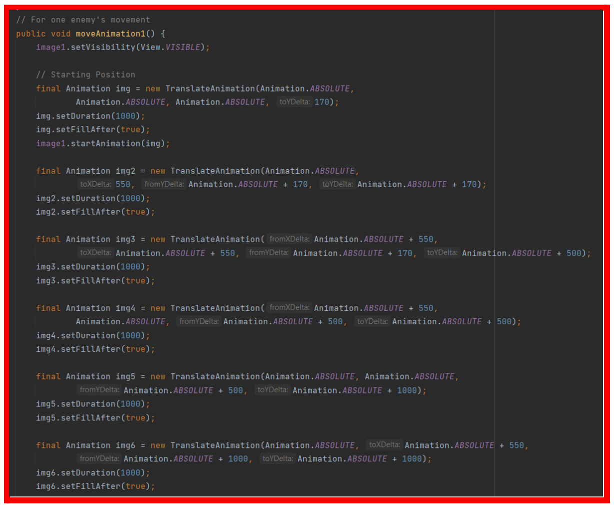


```java
public void moveAnimation(ImageView image) {...}
public void endRound() {...}
```

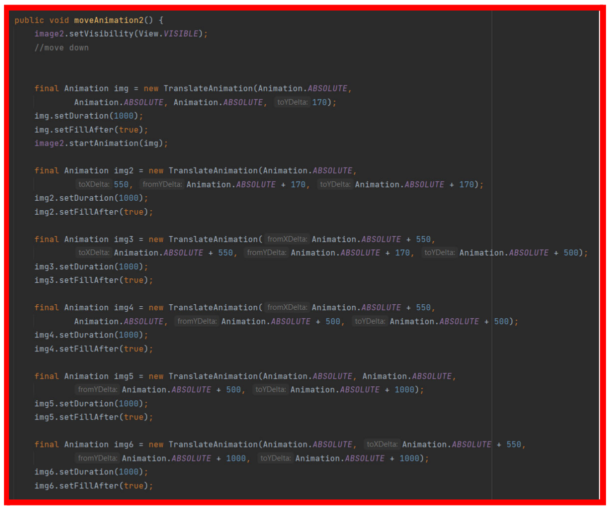
These 3 methods are over 700 lines long. The long methods are because of repeated actions and lines that are copies of one another. In order to fix this, we will remove unused lines of code and shorten the code that exists now so that the only lines are the ones that serve function and are necessary to the code compiling.

This fix allows an ImageView image to pass through rather than each method having its own image variable for each method. It also uses an endRound method to take the place of the endinging method that would have otherwise been in the last 40 lines of moveAnimation1, 2 and 3. This shortens the code by over 400 lines.

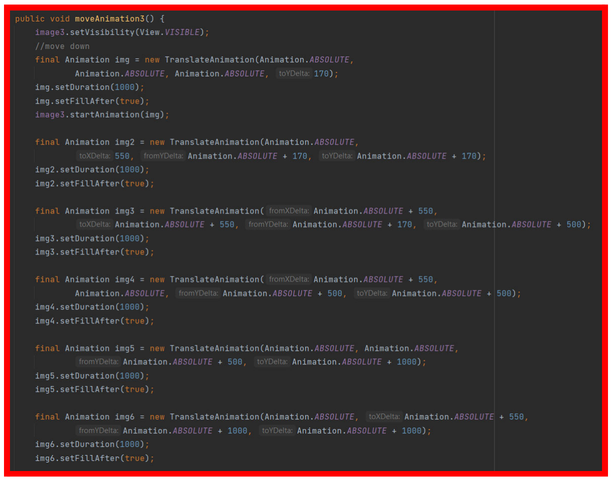
3) Dispensables: Duplicated Code



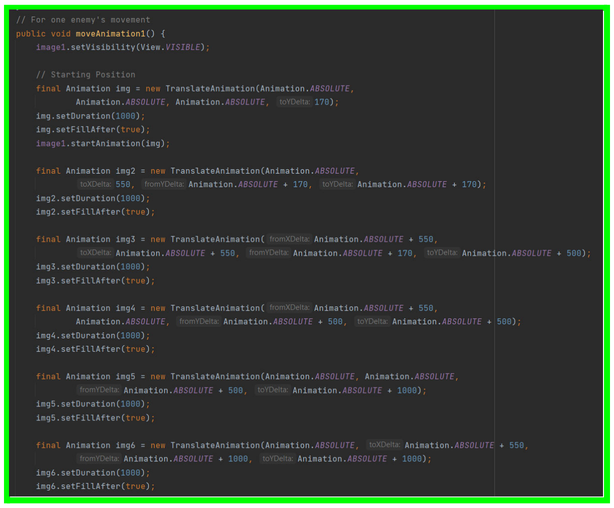
```java
// For one enemy's movement
public void moveAnimation1() {
    image1.setVisibility(View.VISIBLE);

    // Starting Position
    final Animation img = new TranslateAnimation(Animation.ABSOLUTE,
            Animation.ABSOLUTE, Animation.ABSOLUTE,  toYDelta: 170);
    img.setDuration(1000);
    img.setFillAfter(true);
    image1.startAnimation(img);

    final Animation img2 = new TranslateAnimation(Animation.ABSOLUTE,
             toXDelta: 550,  fromYDelta: Animation.ABSOLUTE + 170,  toYDelta: Animation.ABSOLUTE + 170);
    img2.setDuration(1000);
    img2.setFillAfter(true);

    final Animation img3 = new TranslateAnimation( fromXDelta: Animation.ABSOLUTE + 550,
             toXDelta: Animation.ABSOLUTE + 550,  fromYDelta: Animation.ABSOLUTE + 170,  toYDelta: Animation.ABSOLUTE + 500);
    img3.setDuration(1000);
    img3.setFillAfter(true);

    final Animation img4 = new TranslateAnimation( fromXDelta: Animation.ABSOLUTE + 550,
            Animation.ABSOLUTE,  fromYDelta: Animation.ABSOLUTE + 500,  toYDelta: Animation.ABSOLUTE + 500);
    img4.setDuration(1000);
    img4.setFillAfter(true);

    final Animation img5 = new TranslateAnimation(Animation.ABSOLUTE, Animation.ABSOLUTE,
             fromYDelta: Animation.ABSOLUTE + 500,  toYDelta: Animation.ABSOLUTE + 1000);
    img5.setDuration(1000);
    img5.setFillAfter(true);

    final Animation img6 = new TranslateAnimation(Animation.ABSOLUTE,  toXDelta: Animation.ABSOLUTE + 550,
             fromYDelta: Animation.ABSOLUTE + 1000,  toYDelta: Animation.ABSOLUTE + 1000);
    img6.setDuration(1000);
    img6.setFillAfter(true);
```



```java
public void moveAnimation2() {
    image2.setVisibility(View.VISIBLE);
    //move down


    final Animation img = new TranslateAnimation(Animation.ABSOLUTE,
            Animation.ABSOLUTE, Animation.ABSOLUTE,  toYDelta: 170);
    img.setDuration(1000);
    img.setFillAfter(true);
    image2.startAnimation(img);

    final Animation img2 = new TranslateAnimation(Animation.ABSOLUTE,
             toXDelta: 550,  fromYDelta: Animation.ABSOLUTE + 170,  toYDelta: Animation.ABSOLUTE + 170);
    img2.setDuration(1000);
    img2.setFillAfter(true);

    final Animation img3 = new TranslateAnimation( fromXDelta: Animation.ABSOLUTE + 550,
             toXDelta: Animation.ABSOLUTE + 550,  fromYDelta: Animation.ABSOLUTE + 170,  toYDelta: Animation.ABSOLUTE + 500);
    img3.setDuration(1000);
    img3.setFillAfter(true);

    final Animation img4 = new TranslateAnimation( fromXDelta: Animation.ABSOLUTE + 550,
            Animation.ABSOLUTE,  fromYDelta: Animation.ABSOLUTE + 500,  toYDelta: Animation.ABSOLUTE + 500);
    img4.setDuration(1000);
    img4.setFillAfter(true);

    final Animation img5 = new TranslateAnimation(Animation.ABSOLUTE, Animation.ABSOLUTE,
             fromYDelta: Animation.ABSOLUTE + 500,  toYDelta: Animation.ABSOLUTE + 1000);
    img5.setDuration(1000);
    img5.setFillAfter(true);

    final Animation img6 = new TranslateAnimation(Animation.ABSOLUTE,  toXDelta: Animation.ABSOLUTE + 550,
             fromYDelta: Animation.ABSOLUTE + 1000,  toYDelta: Animation.ABSOLUTE + 1000);
    img6.setDuration(1000);
    img6.setFillAfter(true);
```



```java
public void moveAnimation3() {
    image3.setVisibility(View.VISIBLE);
    //move down
    final Animation img = new TranslateAnimation(Animation.ABSOLUTE,
            Animation.ABSOLUTE, Animation.ABSOLUTE,  toYDelta: 170);
    img.setDuration(1000);
    img.setFillAfter(true);
    image3.startAnimation(img);

    final Animation img2 = new TranslateAnimation(Animation.ABSOLUTE,
             toXDelta: 550,  fromYDelta: Animation.ABSOLUTE + 170,  toYDelta: Animation.ABSOLUTE + 170);
    img2.setDuration(1000);
    img2.setFillAfter(true);

    final Animation img3 = new TranslateAnimation( fromXDelta: Animation.ABSOLUTE + 550,
             toXDelta: Animation.ABSOLUTE + 550,  fromYDelta: Animation.ABSOLUTE + 170,  toYDelta: Animation.ABSOLUTE + 500);
    img3.setDuration(1000);
    img3.setFillAfter(true);

    final Animation img4 = new TranslateAnimation( fromXDelta: Animation.ABSOLUTE + 550,
            Animation.ABSOLUTE,  fromYDelta: Animation.ABSOLUTE + 500,  toYDelta: Animation.ABSOLUTE + 500);
    img4.setDuration(1000);
    img4.setFillAfter(true);

    final Animation img5 = new TranslateAnimation(Animation.ABSOLUTE, Animation.ABSOLUTE,
             fromYDelta: Animation.ABSOLUTE + 500,  toYDelta: Animation.ABSOLUTE + 1000);
    img5.setDuration(1000);
    img5.setFillAfter(true);

    final Animation img6 = new TranslateAnimation(Animation.ABSOLUTE,  toXDelta: Animation.ABSOLUTE + 550,
             fromYDelta: Animation.ABSOLUTE + 1000,  toYDelta: Animation.ABSOLUTE + 1000);
    img6.setDuration(1000);
    img6.setFillAfter(true);
```

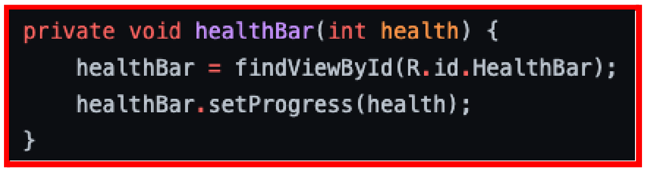


```java
// For one enemy's movement
public void moveAnimation1() {
    image1.setVisibility(View.VISIBLE);

    // Starting Position
    final Animation img = new TranslateAnimation(Animation.ABSOLUTE,
            Animation.ABSOLUTE, Animation.ABSOLUTE,  toYDelta: 170);
    img.setDuration(1000);
    img.setFillAfter(true);
    image1.startAnimation(img);

    final Animation img2 = new TranslateAnimation(Animation.ABSOLUTE,
             toXDelta: 550,  fromYDelta: Animation.ABSOLUTE + 170,  toYDelta: Animation.ABSOLUTE + 170);
    img2.setDuration(1000);
    img2.setFillAfter(true);

    final Animation img3 = new TranslateAnimation( fromXDelta: Animation.ABSOLUTE + 550,
             toXDelta: Animation.ABSOLUTE + 550,  fromYDelta: Animation.ABSOLUTE + 170,  toYDelta: Animation.ABSOLUTE + 500);
    img3.setDuration(1000);
    img3.setFillAfter(true);

    final Animation img4 = new TranslateAnimation( fromXDelta: Animation.ABSOLUTE + 550,
            Animation.ABSOLUTE,  fromYDelta: Animation.ABSOLUTE + 500,  toYDelta: Animation.ABSOLUTE + 500);
    img4.setDuration(1000);
    img4.setFillAfter(true);

    final Animation img5 = new TranslateAnimation(Animation.ABSOLUTE, Animation.ABSOLUTE,
             fromYDelta: Animation.ABSOLUTE + 500,  toYDelta: Animation.ABSOLUTE + 1000);
    img5.setDuration(1000);
    img5.setFillAfter(true);

    final Animation img6 = new TranslateAnimation(Animation.ABSOLUTE,  toXDelta: Animation.ABSOLUTE + 550,
             fromYDelta: Animation.ABSOLUTE + 1000,  toYDelta: Animation.ABSOLUTE + 1000);
    img6.setDuration(1000);
    img6.setFillAfter(true);
```

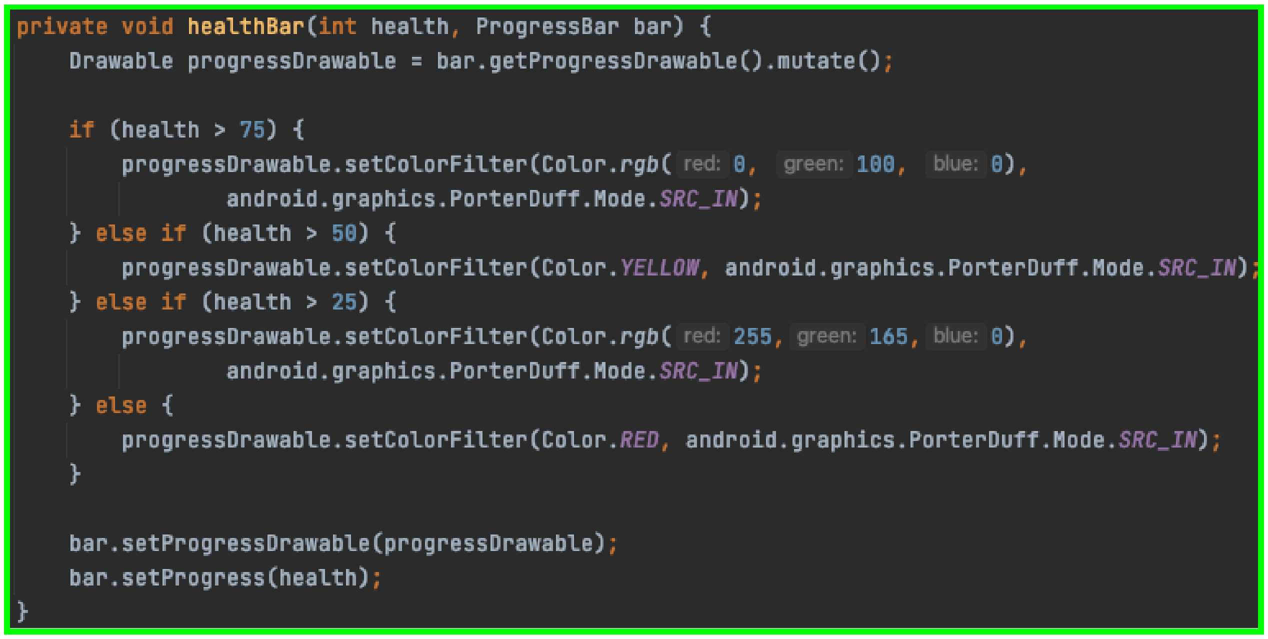
This code is essentially the same with the only exception being the last 40 lines in each method. In order to change this, we make individual methods for the differential code and make the duplicate code the same method. This would allow the methods to be called in the same order, but save hundreds of lines of duplicated code.

This solution eliminates the duplicated code by making methods that represent the duplicated code. Rather than the methods having the same code, move versatile methods will take over the previous code.

4) Dispensables: Duplicated Code



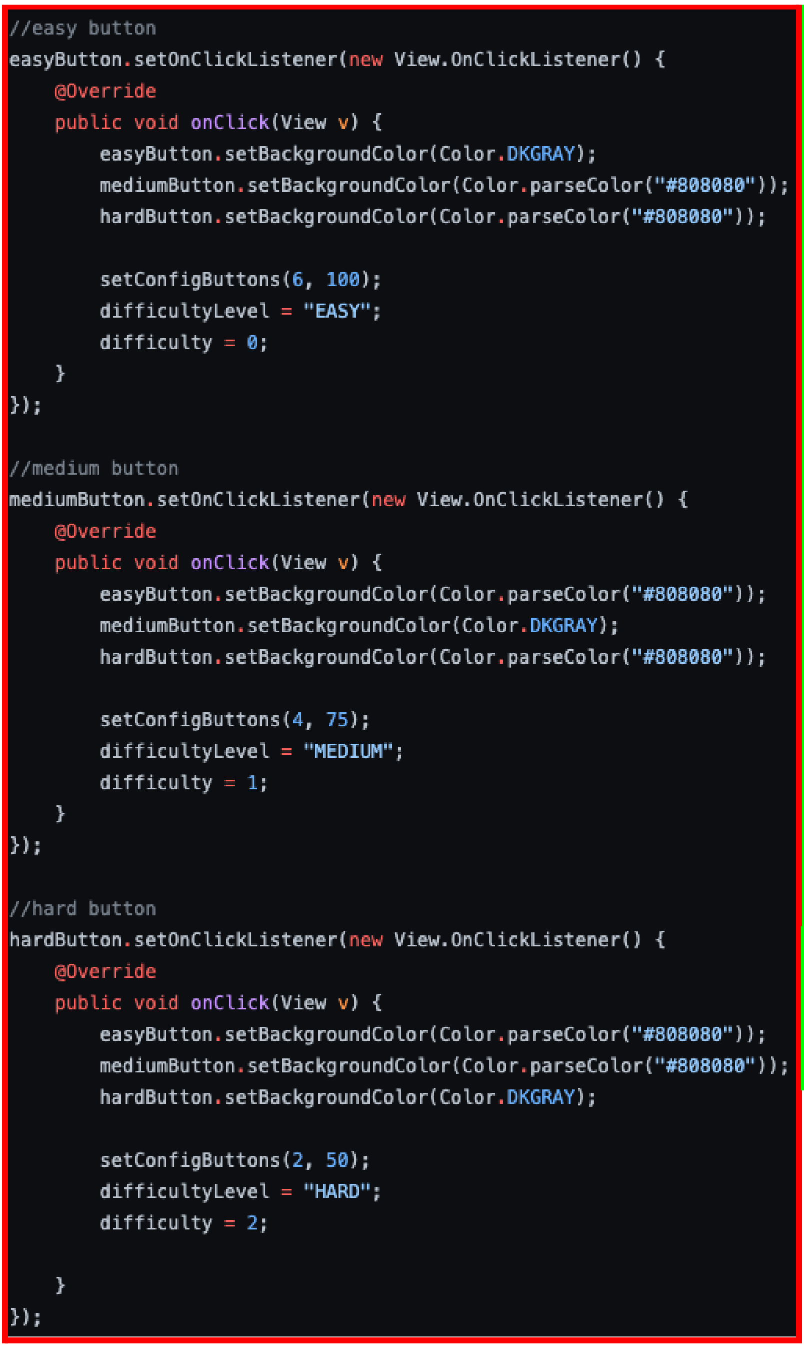
```java
private void healthBar(int health) {
    healthBar = findViewById(R.id.HealthBar);
    healthBar.setProgress(health);
}
```



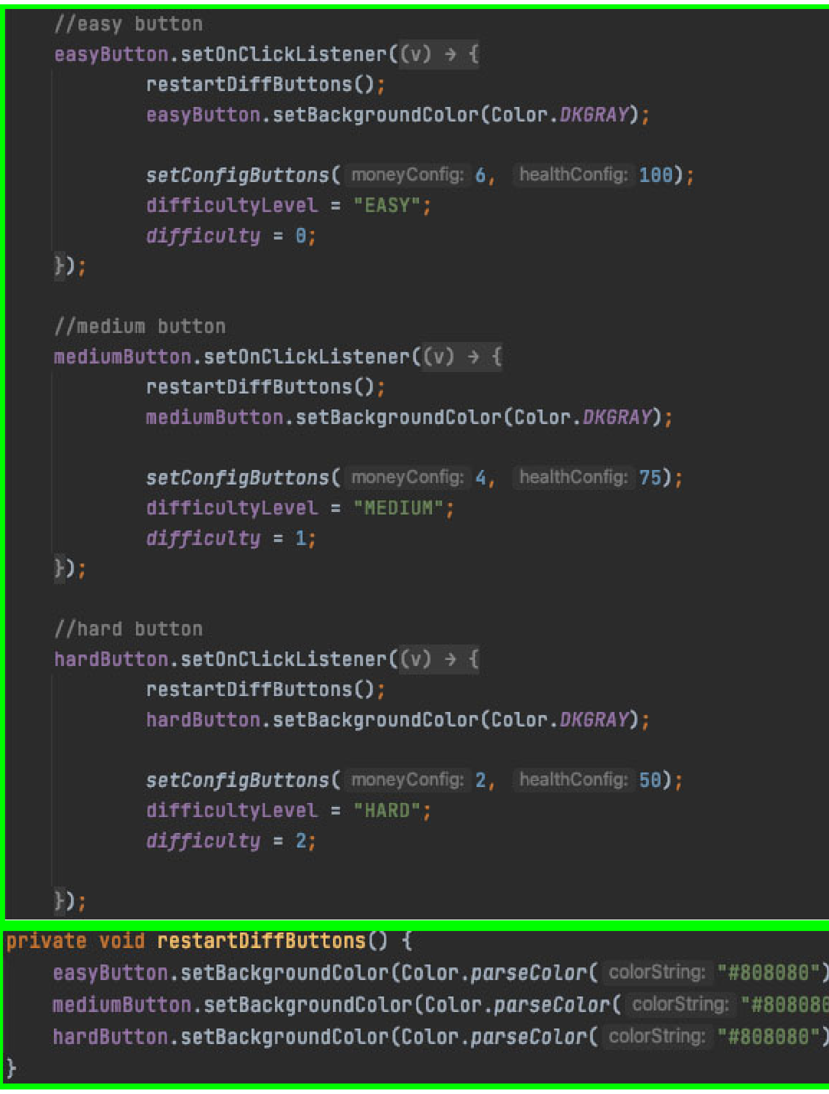
```java
private void healthBar(int health, ProgressBar bar) {
    Drawable progressDrawable = bar.getProgressDrawable().mutate();

    if (health > 75) {
        progressDrawable.setColorFilter(Color.rgb( red: 0,  green: 100,  blue: 0),
                android.graphics.PorterDuff.Mode.SRC_IN);
    } else if (health > 50) {
        progressDrawable.setColorFilter(Color.YELLOW, android.graphics.PorterDuff.Mode.SRC_IN);
    } else if (health > 25) {
        progressDrawable.setColorFilter(Color.rgb( red: 255, green: 165, blue: 0),
                android.graphics.PorterDuff.Mode.SRC_IN);
    } else {
        progressDrawable.setColorFilter(Color.RED, android.graphics.PorterDuff.Mode.SRC_IN);
    }

    bar.setProgressDrawable(progressDrawable);
    bar.setProgress(health);
}
```

The first picture was the original function to update the health bar for the monument. However, when we have multiple health bars for the other towers and enemies, we will need the same function. Because of this, we have adapted the function to work with all progress bars. Along with this, the bar will be a different color based on the health of the object.

5) **Dispensables: Duplicated Code**



```java
//easy button
easyButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        easyButton.setBackgroundColor(Color.DKGRAY);
        mediumButton.setBackgroundColor(Color.parseColor("#808080"));
        hardButton.setBackgroundColor(Color.parseColor("#808080"));

        setConfigButtons(6, 100);
        difficultyLevel = "EASY";
        difficulty = 0;
    }
});

//medium button
mediumButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        easyButton.setBackgroundColor(Color.parseColor("#808080"));
        mediumButton.setBackgroundColor(Color.DKGRAY);
        hardButton.setBackgroundColor(Color.parseColor("#808080"));

        setConfigButtons(4, 75);
        difficultyLevel = "MEDIUM";
        difficulty = 1;
    }
});

//hard button
hardButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        easyButton.setBackgroundColor(Color.parseColor("#808080"));
        mediumButton.setBackgroundColor(Color.parseColor("#808080"));
        hardButton.setBackgroundColor(Color.DKGRAY);

        setConfigButtons(2, 50);
        difficultyLevel = "HARD";
        difficulty = 2;

    }
});
```



```java
    //easy button
    easyButton.setOnClickListener((v) → {
            restartDiffButtons();
            easyButton.setBackgroundColor(Color.DKGRAY);

            setConfigButtons( moneyConfig: 6,  healthConfig: 100);
            difficultyLevel = "EASY";
            difficulty = 0;
    });

    //medium button
    mediumButton.setOnClickListener((v) → {
            restartDiffButtons();
            mediumButton.setBackgroundColor(Color.DKGRAY);

            setConfigButtons( moneyConfig: 4,  healthConfig: 75);
            difficultyLevel = "MEDIUM";
            difficulty = 1;
    });

    //hard button
    hardButton.setOnClickListener((v) → {
            restartDiffButtons();
            hardButton.setBackgroundColor(Color.DKGRAY);

            setConfigButtons( moneyConfig: 2,  healthConfig: 50);
            difficultyLevel = "HARD";
            difficulty = 2;

    });
private void restartDiffButtons() {
    easyButton.setBackgroundColor(Color.parseColor( colorString: "#808080")
    mediumButton.setBackgroundColor(Color.parseColor( colorString: "#80808
    hardButton.setBackgroundColor(Color.parseColor( colorString: "#808080")
}
```

On the right is the code for the configuration screen. When selecting the difficulty, only the button which has been selected should be dark. The way we originally had it is shown on the left. Here we would recolor each button which leads button.setBackgroundColor(Color.DKGRAY) to be written 6 times.

Instead, the function restartDiffButtons() (on the right) changes the background of the button to make it seem like none of the buttons have been selected.