

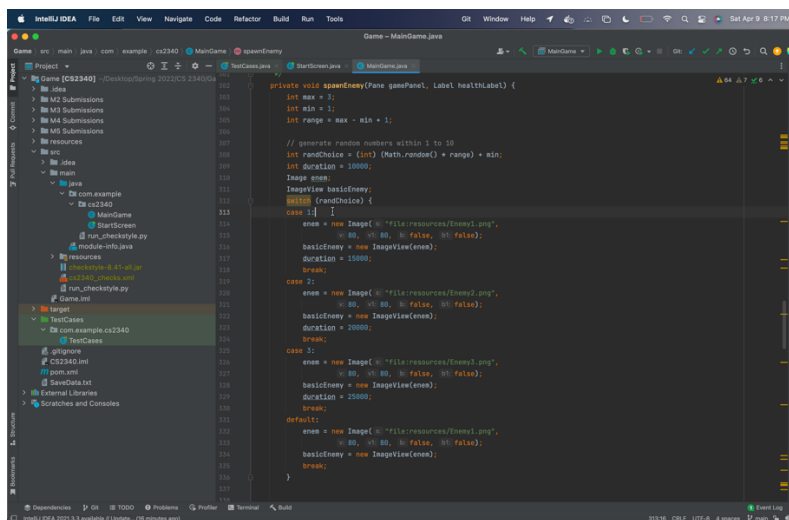
Code Smells M5

Faiyaz

Dispensables - Duplicate Code - Consolidating Conditional Expressions

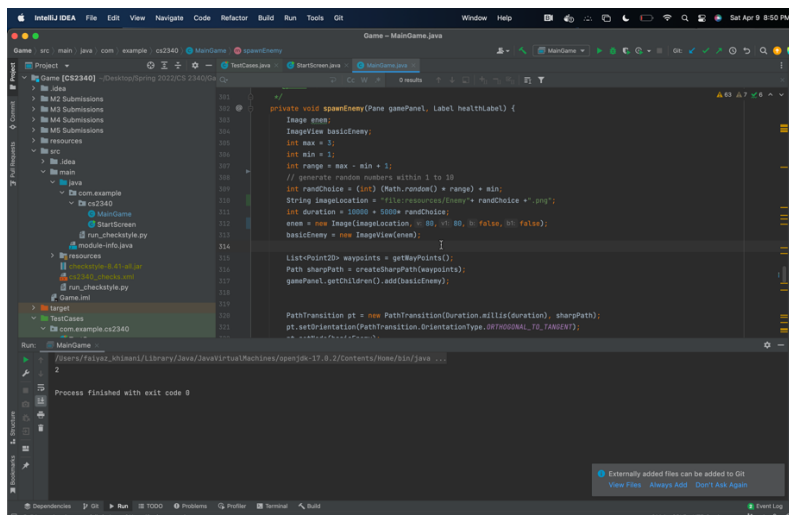
In our spawnEnemy code, we had a switch state that not only made it harder to understand what is going on but also had the same functionality within the statements. With the new changes, it is easier to maintain any changes in the system and make any future updates to enemies spawning.

Before:



```
private void spawnEnemy(Pane gamePanel, Label healthLabel) {  
    int max = 3;  
    int min = 1;  
    int range = max - min + 1;  
  
    // generate random numbers within 1 to 10  
    int randChoice = (int) (Math.random() * range) + min;  
    int duration = 10000;  
    Image enemy;  
    ImageView basicEnemy;  
    switch (randChoice) {  
        case 1:  
            enemy = new Image("file:resources/Enemy1.png",  
                             80, 80, 80, 80, false, 80, false);  
            basicEnemy = new ImageView(enemy);  
            duration = 10000;  
            break;  
        case 2:  
            enemy = new Image("file:resources/Enemy2.png",  
                             80, 80, 80, 80, false, 80, false);  
            basicEnemy = new ImageView(enemy);  
            duration = 10000;  
            break;  
        case 3:  
            enemy = new Image("file:resources/Enemy3.png",  
                             80, 80, 80, 80, false, 80, false);  
            basicEnemy = new ImageView(enemy);  
            duration = 10000;  
            break;  
        default:  
            enemy = new Image("file:resources/Enemy1.png",  
                             80, 80, 80, 80, false, 80, false);  
            basicEnemy = new ImageView(enemy);  
            break;  
    }  
}
```

After:



```
private void spawnEnemy(Pane gamePanel, Label healthLabel) {  
    Image enemy;  
    ImageView basicEnemy;  
    int max = 3;  
    int min = 1;  
    int range = max - min + 1;  
  
    // generate random numbers within 1 to 10  
    int randChoice = (int) (Math.random() * range) + min;  
    String imageLocation = "file:resources/Enemy" + randChoice + ".png";  
    int duration = 10000 + 5000 * randChoice;  
    enemy = new Image(imageLocation, 80, 80, 80, 80, false, 80, false);  
    basicEnemy = new ImageView(enemy);  
    List<Point2D> waypoints = getWayPoints();  
    Path sharpPath = createSharpPath(waypoints);  
    gamePanel.getChildren().add(basicEnemy);  
  
    PathTransition pt = new PathTransition(Duration.millis(duration), sharpPath);  
    pt.setOrientation(PathTransition.OrientationType.ROTATIONAL_75_PERCENT);  
    pt.playFromStart();  
}
```

Shammo

Change Preventers – Shotgun Surgery

While determining collision logistics for our objects, oftentimes, we'd perform these collisions through a direct manipulation of our object's image. For example, to find the position of an enemy, we'd use the position of its sprite. With the new changes, classes have been implemented to hold all the attributes of our objects together so as to prevent large quantities of changes when altering objects. Now, a change to the class itself can prevent the need for large volumes of changes.

Before:

```
for (Enemy enemy: activeEnemies) {  
    if (checkIfInRange(tower.getLayoutX(), tower.getLayoutY(), enemy.getX(), enemy.getY(), range: 200)) {  
        if (tower.getImage().getPixelReader().getArgb(0, 0) == longShotPic.getPixelReader().getArgb(0, 0)) {  
            spawnProjectileLongShot(gamePanel, tower.getLayoutX(), tower.getLayoutY(), tower.getRotate());  
        } else if (tower.getImage().getPixelReader().getArgb(0, 0) == multiShotPic.getPixelReader().getArgb(0, 0)) {  
            spawnProjectileMultiShot(gamePanel, tower.getLayoutX(), tower.getLayoutY());  
        } else if (tower.getImage().getPixelReader().getArgb(0, 0) == spreadShotPic.getPixelReader().getArgb(0, 0)) {  
            spawnProjectileSpreadShot(gamePanel, tower.getLayoutX(), tower.getLayoutY(), tower.getRotate());  
        }  
        break;  
    }  
}
```

The code above referenced a list of enemy images rather than enemy objects themselves

After:

```
public class Enemy {
    ImageView img;
    Double x;
    Double y;
    Integer health = 3;
    Rectangle healthBar;

    public Enemy (ImageView image, Pane gamePanel) {
        this.x = -50.0;
        this.y = -50.0;
        this.img = image;
        this.healthBar = new Rectangle(this.x, v1: this.y - 10, (this.health * 27), v3: 5);
        this.healthBar.setFill(Color.GREEN);
        gamePanel.getChildren().add(healthBar);
    }

    public void setX(Double x) { this.x = x; }
    public void setY(Double y) { this.y = y; }

    public Double getX() { return this.x; }
    public Double getY() { return this.y; }

    public ImageView getImage() { return this.img; }

    public void updateHealth(Pane gamePanel) {
        this.healthBar.setX(this.x);
        this.healthBar.setY(this.y - 10);
        this.healthBar.setWidth(this.health * 27);
    }
}

private void spawnEnemy(Pane gamePanel, Label healthLabel) {
```

An enemy container has been made

Gowrav

Dispensibles – Duplicated Code

Some of our projectile functionality methods featured a handful of duplicated and repetitive code, one example being determining the orientation of a tower for a selector variable. In the `spawnProjectileLongShot` and `spawnProjectileSpreadShot` methods, there were unnecessary switch statements or long if-else statements to determine the selector variable. The simplest solution was just to create an equation.

Before

```
private void spawnProjectileLongShot(Pane gamePanel, double towerX, double towerY, double towerRotate) {  
  
    Image bulletPic = new Image(s: "file:resources/Bullet.png",  
                                v: 20, v1: 20, b: false, b1: false);  
    ImageView bullet = new ImageView(bulletPic);  
  
    List<Point2D> waypoints = new ArrayList<>();  
  
    double[] xAdd = {300, 0, -300, 0};  
    double[] yAdd = {0, 300, 0, -300};  
  
    int selector = 0;  
    if (towerRotate == 0) {  
        selector = 0;  
    }  
    if (towerRotate == 90) {  
        selector = 1;  
    }  
    if (towerRotate == 180) {  
        selector = 2;  
    }  
    if (towerRotate == 270) {  
        selector = 3;  
    }  
}
```

After

```
private void spawnProjectileSpreadShot(Pane gamePanel, double towerX, double towerY, double towerRotate) {  
  
    double[] xAdd = {300, 0, -300, 0};  
    double[] yAdd = {0, 300, 0, -300};  
  
    double[] offSetX = {0, -20, 20};  
    double[] offSetY = {0, -20, 20};  
    int selector = (int) towerRotate / 90;  
}
```

By casting the result of “towerRotate / 90” as an int, we don’t longer need a switch statement or multiple if-else statements.

Eric

Bloater - Long method

After noticing that our `updateScene()` method was entirely too long, we decided the first step in addressing it was turning some of its parts into separate methods. For example, the dozen or so lines used to generate money could be removed from `updateScene()` and turned into its own separate method, like so.

Before:

```
@Override
public void handle(ActionEvent event) {
    gamePane = gamePanel;
    Timeline moneyPerSecond = new Timeline(
        new KeyFrame(Duration.millis(100 * gameDifficulty),
            new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    money += 1;
                    moneyLabel.setText(Integer.toString(money));
                }
            })
    );
    moneyPerSecond.setCycleCount(Animation.INDEFINITE); // loop forever
    moneyPerSecond.play();
}
```

The logic for money generation was inside of another method, when it could and should have been its own method.

After:

```
public void startMoneyGeneration(Label moneyLabel) {
    Timeline moneyPerSecond = new Timeline(
        new KeyFrame(Duration.millis(100 * gameDifficulty),
            new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    if (inCombat) {
                        money += 1;
                        moneyLabel.setText(Integer.toString(money));
                    }
                }
            })
    );
    moneyPerSecond.setCycleCount(Animation.INDEFINITE); // loop forever
    moneyPerSecond.play();
}
```

The long method has been shortened, and the money generation logic has been separated.