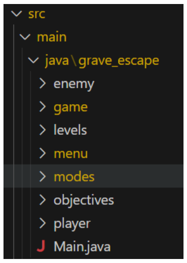
Overview

In this report, we have included and documented the smells that we have identified and the solutions for addressing the detected smells as well as the performed refactoring included with images. Our github activity shows the commits that we have made to make the appropriate changes to our code.

Code Smells

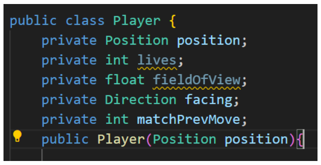
1. Badly Structured Project (All files)
2. Data Clumps (Player.java)
3. Large Class (Enemy.java)
4. Large Class (Player.java)
5. Badly Structured Project (Levels inside levels)
6. Badly Structured Project (Door.java & Wall.java)
7. Duplicate Code (GameFactory.java)
8. Duplicate Code (Grid.java, Level.java & GamePanel.java)

Refactoring Techniques to Solve the Code Smells



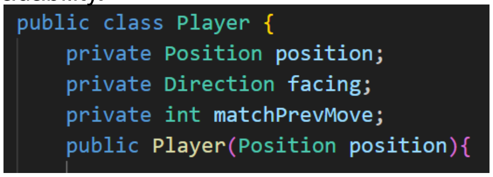
```
src
  main
    java\grave_escape
      enemy
      game
      levels
      menu
      modes
      objectives
      player
      Main.java
```

1. Organized Files
   - Made folders for each important component of our game rather than having all the files inside the java folder. This helps with improved readability and appropriate grouping.
2. Extract Class/Delete data
   - The class had initializations of variables that were not used so we moved/deleted unnecessary variables.This helps with the fact that we now have less lines of unnecessary code in our files for improved readability.



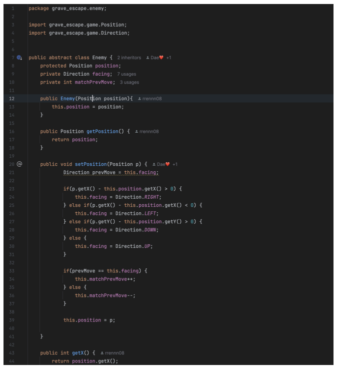
```java
public class Player {
    private Position position;
    private int lives;
    private float fieldOfView;
    private Direction facing;
    private int matchPrevMove;
    public Player(Position position){
```

→



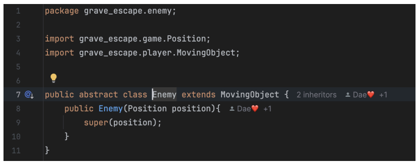
```java
public class Player {
    private Position position;
    private Direction facing;
    private int matchPrevMove;
    public Player(Position position){
```

3. Extract Class
   - Moving over data and methods that have to do with getting the position of and Enemy Tracking to a MovingObject.java file. Removed method (code) duplication in the Enemy class.

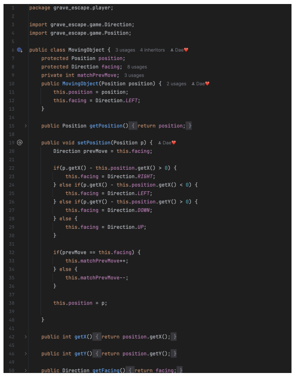


```java
package grave_escape.enemy;

import grave_escape.game.Position;
import grave_escape.game.Direction;


public abstract class Enemy {
    protected Position position;
    private Direction facing;
    private int matchPrevMove;

    public Enemy(Position position){
        this.position = position;
    }

    public Position getPosition() {
        return position;
    }

    public void setPosition(Position p) {
            Direction prevMove = this.facing;

            if(p.getX() - this.position.getX() > 0) {
                this.facing = Direction.RIGHT;
            } else if(p.getX() - this.position.getX() < 0) {
                this.facing = Direction.LEFT;
            } else if(p.getY() - this.position.getY() > 0) {
                this.facing = Direction.DOWN;
            } else {
                this.facing = Direction.UP;
            }

            if(prevMove == this.facing) {
                this.matchPrevMove++;
            } else {
                this.matchPrevMove--;
            }

            this.position = p;

    }

    public int getX() {
        return position.getX();
```

→



```java
package grave_escape.enemy;

import grave_escape.game.Position;
import grave_escape.player.MovingObject;


public abstract class Enemy extends MovingObject {
    public Enemy(Position position){
        super(position);
    }
}
```

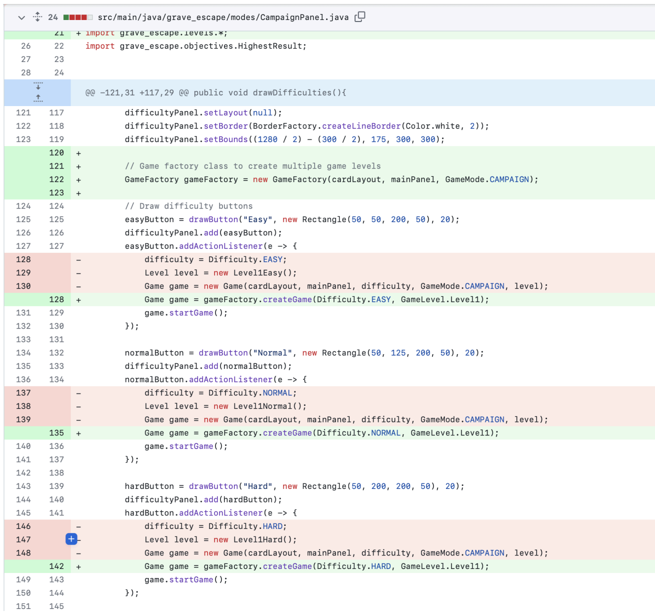


```java
package grave_escape.player;

import grave_escape.game.Direction;
import grave_escape.game.Position;

public class MovingObject {
    protected Position position;
    protected Direction facing;
    private int matchPrevMove;
    public MovingObject(Position position) {
        this.position = position;
        this.facing = Direction.LEFT;
    }

    public Position getPosition() { return position; }

    public void setPosition(Position p) {
        Direction prevMove = this.facing;

        if(p.getX() - this.position.getX() > 0) {
            this.facing = Direction.RIGHT;
        } else if(p.getX() - this.position.getX() < 0) {
            this.facing = Direction.LEFT;
        } else if(p.getY() - this.position.getY() > 0) {
            this.facing = Direction.DOWN;
        } else {
            this.facing = Direction.UP;
        }

        if(prevMove == this.facing) {
            this.matchPrevMove++;
        } else {
            this.matchPrevMove--;
        }

        this.position = p;

    }

    public int getX() { return position.getX(); }

    public int getY() { return position.getY(); }

    public Direction getFacing() { return facing; }
```

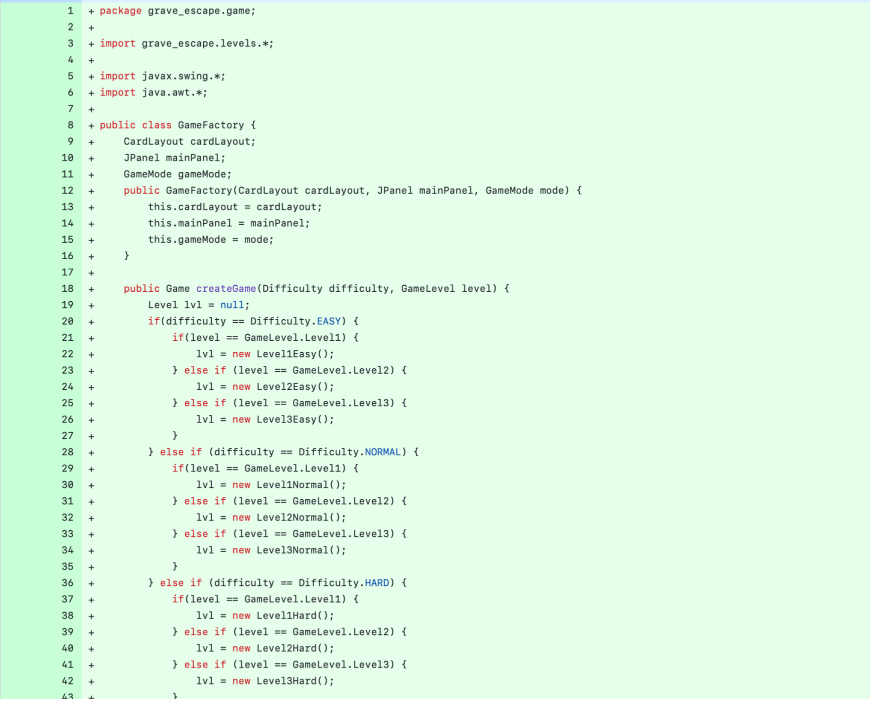
4. Extract Class
   - Moving over data and methods that have to do with getting the position of the player to a MovingObject.java file. Removed method (code) duplication in the Player class.
5. Organized Files
   - Added subfolders to the levels folder to specify the levels for each corresponding difficulty/progression. Having each level grouped makes it easier to make changes as well as improved readability.
6. Organized Files
   - Moved over the Door.java and Wall.java to a different “structure” folder since they did not really fit the “criteria” of the objectives folder. The objectives folder now only is left with the HighestResult.java and the Objective.java files. This helps for improved readability as well as more appropriate grouping.

7. Refactor to Factory Pattern

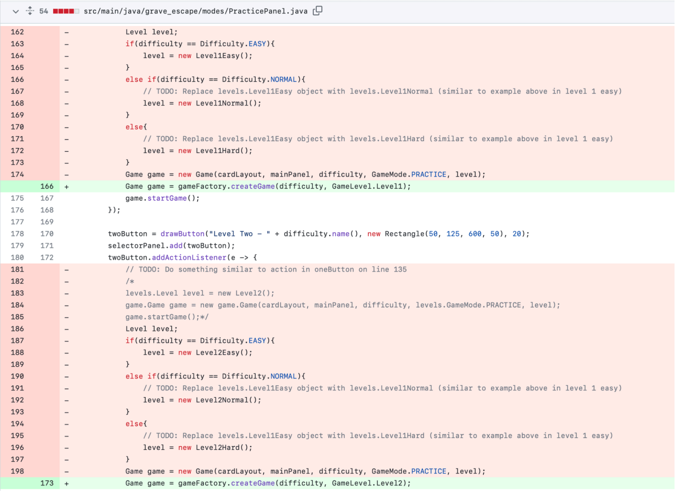
- Extracted logic for creating game levels and difficulties into a new GameFactory class.
- Consolidated Level object creation (e.g., LevelEasy, LevelNormal, etc.) based on difficulty and game level.
- Removed duplicate code across buttons by using GameFactory.createGame() to encapsulate game initialization logic.



```
24 src/main/java/grave_escape/modes/CampaignPanel.java
+ import grave_escape.levels.*;
  import grave_escape.objectives.HighestResult;


@@ -121,31 +117,29 @@ public void drawDifficulties(){
          difficultyPanel.setLayout(null);
          difficultyPanel.setBorder(BorderFactory.createLineBorder(Color.white, 2));
          difficultyPanel.setBounds((1280 / 2) - (300 / 2), 175, 300, 300);
+
+         // Game factory class to create multiple game levels
+         GameFactory gameFactory = new GameFactory(cardLayout, mainPanel, GameMode.CAMPAIGN);
+
          // Draw difficulty buttons
          easyButton = drawButton("Easy", new Rectangle(50, 50, 200, 50), 20);
          difficultyPanel.add(easyButton);
          easyButton.addActionListener(e -> {
-             difficulty = Difficulty.EASY;
-             Level level = new Level1Easy();
-             Game game = new Game(cardLayout, mainPanel, difficulty, GameMode.CAMPAIGN, level);
+             Game game = gameFactory.createGame(Difficulty.EASY, GameLevel.Level1);
              game.startGame();
          });

          normalButton = drawButton("Normal", new Rectangle(50, 125, 200, 50), 20);
          difficultyPanel.add(normalButton);
          normalButton.addActionListener(e -> {
-             difficulty = Difficulty.NORMAL;
-             Level level = new Level1Normal();
-             Game game = new Game(cardLayout, mainPanel, difficulty, GameMode.CAMPAIGN, level);
+             Game game = gameFactory.createGame(Difficulty.NORMAL, GameLevel.Level1);
              game.startGame();
          });

          hardButton = drawButton("Hard", new Rectangle(50, 200, 200, 50), 20);
          difficultyPanel.add(hardButton);
          hardButton.addActionListener(e -> {
-             difficulty = Difficulty.HARD;
-             Level level = new Level1Hard();
-             Game game = new Game(cardLayout, mainPanel, difficulty, GameMode.CAMPAIGN, level);
+             Game game = gameFactory.createGame(Difficulty.HARD, GameLevel.Level1);
              game.startGame();
          });

```



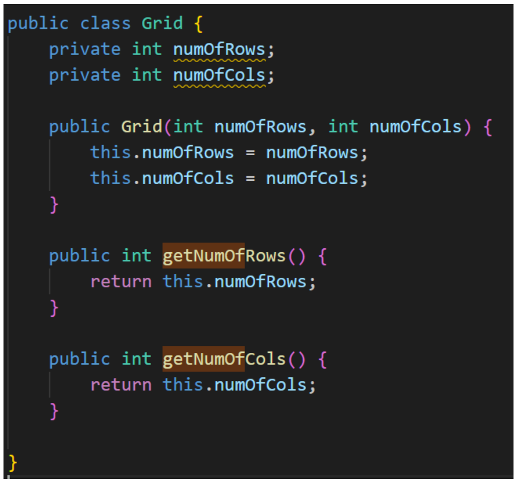
```
+ package grave_escape.game;
+
+ import grave_escape.levels.*;
+
+ import javax.swing.*;
+ import java.awt.*;
+
+ public class GameFactory {
+     CardLayout cardLayout;
+     JPanel mainPanel;
+     GameMode gameMode;
+     public GameFactory(CardLayout cardLayout, JPanel mainPanel, GameMode mode) {
+         this.cardLayout = cardLayout;
+         this.mainPanel = mainPanel;
+         this.gameMode = mode;
+     }
+
+     public Game createGame(Difficulty difficulty, GameLevel level) {
+         Level lvl = null;
+         if(difficulty == Difficulty.EASY) {
+             if(level == GameLevel.Level1) {
+                 lvl = new Level1Easy();
+             } else if (level == GameLevel.Level2) {
+                 lvl = new Level2Easy();
+             } else if (level == GameLevel.Level3) {
+                 lvl = new Level3Easy();
+             }
+         } else if (difficulty == Difficulty.NORMAL) {
+             if(level == GameLevel.Level1) {
+                 lvl = new Level1Normal();
+             } else if (level == GameLevel.Level2) {
+                 lvl = new Level2Normal();
+             } else if (level == GameLevel.Level3) {
+                 lvl = new Level3Normal();
+             }
+         } else if (difficulty == Difficulty.HARD) {
+             if(level == GameLevel.Level1) {
+                 lvl = new Level1Hard();
+             } else if (level == GameLevel.Level2) {
+                 lvl = new Level2Hard();
+             } else if (level == GameLevel.Level3) {
+                 lvl = new Level3Hard();
+             }
```



```
54 src/main/java/grave_escape/modes/PracticePanel.java
-             Level level;
-             if(difficulty == Difficulty.EASY){
-                 level = new Level1Easy();
-             }
-             else if(difficulty == Difficulty.NORMAL){
-                 // TODO: Replace levels.Level1Easy object with levels.Level1Normal (similar to example above in level 1 easy)
-                 level = new Level1Normal();
-             }
-             else{
-                 // TODO: Replace levels.Level1Easy object with levels.Level1Hard (similar to example above in level 1 easy)
-                 level = new Level1Hard();
-             }
-             Game game = new Game(cardLayout, mainPanel, difficulty, GameMode.PRACTICE, level);
+             Game game = gameFactory.createGame(difficulty, GameLevel.Level1);
              game.startGame();
          });

          twoButton = drawButton("Level Two - " + difficulty.name(), new Rectangle(50, 125, 600, 50), 20);
          selectorPanel.add(twoButton);
          twoButton.addActionListener(e -> {
-             // TODO: Do something similar to action in oneButton on line 135
-             /*
-             levels.Level level = new Level2();
-             game.Game game = new game.Game(cardLayout, mainPanel, difficulty, levels.GameMode.PRACTICE, level);
-             game.startGame();*/
-             Level level;
-             if(difficulty == Difficulty.EASY){
-                 level = new Level2Easy();
-             }
-             else if(difficulty == Difficulty.NORMAL){
-                 // TODO: Replace levels.Level1Easy object with levels.Level1Normal (similar to example above in level 1 easy)
-                 level = new Level2Normal();
-             }
-             else{
-                 // TODO: Replace levels.Level1Easy object with levels.Level1Hard (similar to example above in level 1 easy)
-                 level = new Level2Hard();
-             }
-             Game game = new Game(cardLayout, mainPanel, difficulty, GameMode.PRACTICE, level);
+             Game game = gameFactory.createGame(difficulty, GameLevel.Level2);
```

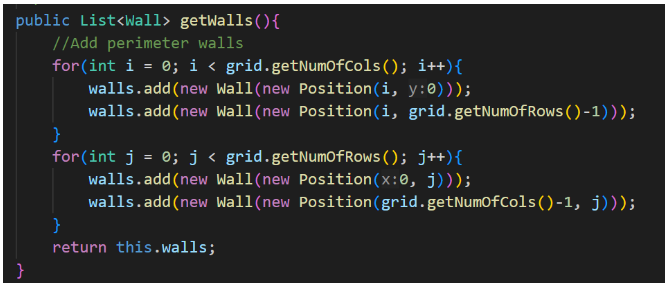
8. Call Method From Different Class

- The numOfRows and numOfCols is used inside Level.java and Gamepanel.java but for better readability, after initializing these variables inside of the Grid class, it would be more efficient to reuse them by just calling this class in the other corresponding files.



```
public class Grid {
    private int numOfRows;
    private int numOfCols;

    public Grid(int numOfRows, int numOfCols) {
        this.numOfRows = numOfRows;
        this.numOfCols = numOfCols;
    }

    public int getNumOfRows() {
        return this.numOfRows;
    }

    public int getNumOfCols() {
        return this.numOfCols;
    }

}
```

→



```
public List<Wall> getWalls(){
    //Add perimeter walls
    for(int i = 0; i < grid.getNumOfCols(); i++){
        walls.add(new Wall(new Position(i, y:0)));
        walls.add(new Wall(new Position(i, grid.getNumOfRows()-1)));
    }
    for(int j = 0; j < grid.getNumOfRows(); j++){
        walls.add(new Wall(new Position(x:0, j)));
        walls.add(new Wall(new Position(grid.getNumOfCols()-1, j)));
    }
    return this.walls;
}
```