**Code Smells**

**Rodrigo Monteiro Suzana 63069**

**(suzana2314)**

**Speculative Genrality**

This smell is located in: *net.sf.freecol.server.generator.ColonizationMapLoader*

Looking specifically at the *ColonizationMapLoader* class, we can pinpoint the speculative class smell, notably in the methods ColonizationMapLoader(File file) throws IOException and Layer loadMap(Game game, Layer layer). This is due to no usage in the code at all, probably it was supposed to be an implementation for future updates. To address this issue, deleting this class would be the right choice.

Bellow is a code snippet of both methods:

public ColonizationMapLoader(File file) throws IOException {

    try (RandomAccessFile reader = new RandomAccessFile(file, "r")) {

        reader.readFully(header);

        int size = header[WIDTH] \* header[HEIGHT];

        layer1 = new byte[size];

        reader.readFully(layer1);

    } catch (EOFException ee) {

        logger.log(Level.SEVERE, "File (" + file + ") is too short.", ee);

    } catch (FileNotFoundException fe) {

        logger.log(Level.SEVERE, "File (" + file + ") was not found.", fe);

    } catch (IOException e) {

        logger.log(Level.SEVERE, "File (" + file + ") is corrupt and cannot be read.", e);

    }

public Layer loadMap(Game game, Layer layer) {

        Specification spec = game.getSpecification();

        Tile[][] tiles = new Tile[header[WIDTH]][header[HEIGHT]];

        Layer highestLayer = layer.compareTo(getHighestLayer()) < 0

            ? layer : getHighestLayer();

        int index = 0;

        TileType tileType = null;

        if (highestLayer == Layer.LAND) {

            // import only the land / water distinction

            for (int y = 0; y < header[HEIGHT]; y++) {

                for (int x = 0; x < header[WIDTH]; x++) {

                    int decimal = layer1[index] & 0xff;

                    int terrain = decimal & 0b11111;

                    tileType = (terrain == OCEAN || terrain == HIGH\_SEAS) ?

                        TileType.WATER : TileType.LAND;

                    index++;

                }

            }

        } else {

            TileImprovementType riverType = spec.getTileImprovementType("model.improvement.river");

            for (int y = 0; y < header[HEIGHT]; y++) {

                for (int x = 0; x < header[WIDTH]; x++) {

                    int decimal = layer1[index] & 0xff;

                    int terrain = decimal & 0b11111;

                    int overlay = decimal >> 5;

                    if (terrain < tiletypes.length) {

                        tileType = spec.getTileType("model.tile." + tiletypes[terrain]);

                    } else if (overlay == 1 || overlay == 3) {

                        tileType = spec.getTileType("model.tile.hills");

                    } else if (overlay == 5 || overlay == 7) {

                        tileType = spec.getTileType("model.tile.mountains");

                    }

                    tiles[x][y] = new Tile(game, tileType, x, y);

                    if (highestLayer == Layer.RIVERS

                        && (overlay == 2 || overlay == 3 || overlay == 6 || overlay == 7)) {

                        TileItemContainer container = new TileItemContainer(game, tiles[x][y]);

                        TileImprovement river =

                            new TileImprovement(game, tiles[x][y], riverType,

                                TileImprovementStyle.getInstance(TileImprovement.EMPTY\_RIVER\_STYLE));//TODO: connections!

                        river.setMagnitude(overlay <= 3 ? 1 : 2);

                        container.tryAddTileItem(river);

                        tiles[x][y].setTileItemContainer(container);

                    }

                    index++;

                }

            }

        }

        return highestLayer;

    }

**Long methods**

This smell is located in: *net.sf.freecol.server.generator.ColonizationMapLoader*

Moreover, within the same class (*ColonizationMapLoader*), there's another code smell: long methods, particularly in the method Layer loadMap(Game game, Layer layer). As the name suggests this method is really long, the nested if-else conditions based on "overlay" and "terrain" might become overly complex and challenging to maintain as the code grows, also because this method isn’t really finished because there is a TODO statement. Extracting some of this logic into separate methods or even utilizing switch cases could significantly enhance readability.

Bellow is a code snippet of the method in question:

public Layer loadMap(Game game, Layer layer) {

        Specification spec = game.getSpecification();

        Tile[][] tiles = new Tile[header[WIDTH]][header[HEIGHT]];

        Layer highestLayer = layer.compareTo(getHighestLayer()) < 0

            ? layer : getHighestLayer();

        int index = 0;

        TileType tileType = null;

        if (highestLayer == Layer.LAND) {

            // import only the land / water distinction

            for (int y = 0; y < header[HEIGHT]; y++) {

                for (int x = 0; x < header[WIDTH]; x++) {

                    int decimal = layer1[index] & 0xff;

                    int terrain = decimal & 0b11111;

                    tileType = (terrain == OCEAN || terrain == HIGH\_SEAS) ?

                        TileType.WATER : TileType.LAND;

                    index++;

                }

            }

        } else {

            TileImprovementType riverType = spec.getTileImprovementType("model.improvement.river");

            for (int y = 0; y < header[HEIGHT]; y++) {

                for (int x = 0; x < header[WIDTH]; x++) {

                    int decimal = layer1[index] & 0xff;

                    int terrain = decimal & 0b11111;

                    int overlay = decimal >> 5;

                    if (terrain < tiletypes.length) {

                        tileType = spec.getTileType("model.tile." + tiletypes[terrain]);

                    } else if (overlay == 1 || overlay == 3) {

                        tileType = spec.getTileType("model.tile.hills");

                    } else if (overlay == 5 || overlay == 7) {

                        tileType = spec.getTileType("model.tile.mountains");

                    }

                    tiles[x][y] = new Tile(game, tileType, x, y);

                    if (highestLayer == Layer.RIVERS

                        && (overlay == 2 || overlay == 3 || overlay == 6 || overlay == 7)) {

                        TileItemContainer container = new TileItemContainer(game, tiles[x][y]);

                        TileImprovement river =

                            new TileImprovement(game, tiles[x][y], riverType,

                                TileImprovementStyle.getInstance(TileImprovement.EMPTY\_RIVER\_STYLE));//TODO: connections!

                        river.setMagnitude(overlay <= 3 ? 1 : 2);

                        container.tryAddTileItem(river);

                        tiles[x][y].setTileItemContainer(container);

                    }

                    index++;

                }

            }

        }

        return highestLayer;

    }

**Data Class**

This smell is located in: *net.sf.freecol.common.model.CombatModel.CombatOdds*

This class is associated as a *data class* code smell primarily because it acts as a container for data with minimal functionality. It only stores a single value (win) and doesn't have any additional methods or behaviors associated with it.To address this, a simple solution involves moving the only variable to another class, or alternatively, either removing the class or adding more functionality to it.

Bellow is a code snippet of the class in question:

public static class CombatOdds {

        public static final double UNKNOWN\_ODDS = -1.0;

        public final double win;

        public CombatOdds(double win) {

            this.win = win;

        }

    }