Exercise 2

Group members: Daniel Langbein, Artem Semenovykh, Sam Tadjiky

1 Duplicated Code

In BoulderAndDiamondController we added a helper method called `getElementSpriteName` to replace the duplicated code fragments.

In RockfordModel we created a private method called `resetState` to reset a list of fields that all duplicated code fragments were changing. After executing this private function we set the fields that were specific to their method to the correct value.

In GroundView we replaced the duplicate code with a method called `drawBackground`. As this method was called in both, the `if` and the `else` case, we moved it out of the if-else construct.

2 PMD

a)

Here's the list of violations. Smells from the lecture are marked green. Indicators of Smells are marked blue.

1. AvoidCatchingGenericException
2. AvoidDeeplyNestedIfStmts → Indicator for too long method, because deeply nested statements are extremely likely to create long methods
3. CognitiveComplexity → Indicator for too long method/class, because using many if-else,for, while, etc. blocks leads to long methods
4. CollapsibleIfStatements
5. CyclomaticComplexity → Indicator for too long method
6. DataClass → Lazy Class
7. ExcessivePublicCount
8. FinalFieldCouldBeStatic
9. GodClass → God Class
10. ImmutableField
11. LawOfDemeter → Inappropriate Intimacy
12. MutableStaticState
13. SimplifyBooleanExpressions
14. SimplifyBooleanReturns
15. SingularField

16. TooManyFields → Indicator for God Class as the class will probably too long if it has too many fields.
17. TooManyMethods → Indicator for God Class as the class will probably too long if it has too any methods.
18. UseUtilityClass

b)

`RockfordModel` does not violate the “TooManyFields” rule. We assumed this was a typo on the homework sheet. Thus we looked at `LevelModel` instead.

We grouped some semantically related fields together. For example there is now one object of type `Point` consisting of the X and Y positions of the cursor instead of two individual fields. Furthermore we converted one field that was only used once to a local variable inside a function.

c)

The field levelModelForGame is a temporary field code smell. It’s initialized and passed to other methods, but never actually used in NavigationBetweenController itself.
We fixed it by converting the temporary field to a local variable.

The field levelEditorController is not just used to store and pass along data, but its methods are actually being called.

3 LevelLoadHelper

Code smells for this class are (not a complete list of course):

1 - Large Class (cross-class smell)

The LevelLoadHelper class is quite long and has multiple responsibilities:

1. Rockford Management - stores and provides access to Rockfords position and instance (`getRockfordPositionX, setRockfordPositionX, setRockfordInstance`, etc)
2. Level Data Loading and Parsing - it loads the level data into instance data space and parses XML file (`loadLevelData, parseLevelData, processNameElement, processDateElement, processSizeElement`)
3. Level Data Access - stores and provides access to level metadata: name, creation date, modification date. (`setLevelId, getLevelId, setDateModifiedValue, getDateModifiedValue`, etc)
4. Grid Management - Creates and manages an array (groundGrid) representing the game level (`constructGridElement, processGridElement`)
5. etc.

This makes the class hard to maintain.
To deal with it, we could consider breaking it down into several smaller, more focused classes.

2 - Unused code (in-class smell)

There are several private setter methods that are never used within the class (e.g., `setNameValue(), setDateCreatedValue(), setDateModifiedValue()`).There are also several imported classes that are not used in the code (e.g., `ExpandingWallModel, EmptyModel, BrickWallModel`, etc.). These could be removed if they're truly unnecessary.

3 - Temporary fields (in-class smell)

There are temporary fields in this class, only used for a specific operation and are not relevant to the object's overall state for most of its lifetime.

- levelDOM: This field is only used during the loading process and becomes unnecessary after the level data is processed.
- xpathBuilder: This field is initialized in `loadLevelData()` and used only during the processing of level data. It's not used after that.
- dateFormatter: This is initialized in the constructor but only used in the `processDateElement()` method.

To deal with it, we can use method local variables instead of instance variables for these temporary fields

4 - Large Method (in-class smell)

The `processGridElement()` method is a good example of a Large Method. It's long, complex, and performs multiple tasks including XML parsing, object creation, and state management.
We should refactor this method and break it down into smaller, more focused methods. For example:

- Extract the XML parsing logic into separate methods.
- Create a method for handling sprite elements.

5 - Inappropriate Intimacy (cross-class smell)

The `LevelLoadHelper` class seems to know too much about the internal structure of `RockfordModel`. Examples:

- The `LevelLoadHelper` class directly sets Rockford's position and instance
- `LevelLoadHelper` knows about Rockford's specific attributes like position, which should ideally be encapsulated within the `RockfordModel` class
- Both classes share the responsibility of managing Rockford's state. `LevelLoadHelper` sets the initial state, while `RockfordModel` manages the ongoing state. This shared responsibility can lead to confusion and maintenance issues.

Code should be refactored. For example, we can encapsulate Rockford's position within the `RockfordModel` class, providing methods to update and retrieve the position rather than exposing it directly.