Exercise 2 – Software Metrics (45 pts)

1) ***Use inCode to compute software metrics on your project, then upload the resulting analysis file to your git repository. Write in the exaplantion PDF file where the analysis file is located. (3 pts)***

The analysis file is located in the same folder as this pdf and is called: "main_1444849845768.result".

2) ***Consider the 'System summary' view regarding the analysis of your project. You can see the design flaws that seem affect your system. Pick the first three design flaws (in order of severity) that affect your software, and for each flaw complete the following points:***

The exercises 'a' and 'b' are repeated 3 times. Once for each design flaw.
The three design flaws are:
1) Schizophrenic class: "GameController"
2) Sibling duplication: "Line" and "Projectile"
3) Data class: "MainMenuController" and "YouLostController"

a.1) ***Explain the design choices or errors leading to the detected design flaw (4pts)***
Schizophrenic class: "GameController"
"GameController" is a schizophrenic class because there is a disjoint set of methods which gets called by a disjoint set of classes. For example the "GameLoop" class is the only class which calls the "addLife", "died", "levelCompleted", "makeCircle", "makeImage", "makeLine" and "updateTime" methods and those methods don't use any variable or other methods of the "GameController" class. Because of this it seems to inCode as if the "GameController" class exists out of methods which don't belong together and which should be split into multiple separate classes.

b.1) ***Fix the design flaw or extensively and precisely explain why this detected flaw is not an error and, thus, should not be fixed (10 pts)***
Under normal circumstances, this design flaw should definitely be solved, but in the case of "GameController" this is not the case. To explain the reasoning behind this decision we first have to explain what the "GameController" class actually is and what it does:
The "GameController" is a controller class of a javaFX view. The view has been designed with the scene editor, which creates an fxml file. The view contains all sorts of objects, for example lines, image and textboxes, and some of those objects have to be controller from the code. To make this possible you can set a controller class for an fxml file and link objects in the fxml file with objects in the controller class. When the fxml file is loaded (meaning a new view has been loaded), the javaFX loader will create the controller class of the fxml file and link the objects in the view with the variables in the controller class. This basically means that the controller class of a view can be used to access objects in the view. You could for example add text to a textbox in the controller class of a view.
As described in exercise 'a.1', the 'flaw' detected by inCode basically means that the methods in the "GameController" class don't seem to belong together in a class because they don't work together. This is not the case however, because all of the methods in the "GameController" class are used to manipulate the same view. Each method might access a different part of the view, but all of those different parts still belong to the same view and this view should be controlled from one place. Splitting the controller class would also go against the javaFX design principals, because fxml files don't support the use of multiple controller classes.
The reason inCode detects this as a flaw, while it isn't, is that the linking of the fxml file objects to controller class variables happens when the fxml file is loaded and this can't be detected by inCode.

a.2) ***Explain the design choices or errors leading to the detected design flaw (4pts)***
Sibling duplication: “Line” and “Projectile”
This design flaw means that a part of the code in the “Line” class is exactly the same as in the “Projectile” class. This is bad practice because code duplication means the code will be harder to maintain: you will have to test the same code multiple times and when there is an error, you will have to fix it multiple times (or when for example implementing a better algorithm, you will have to implement it multiple times).
The code which is duplicated is the code which calculates the closest intersection point between a point and the projectile/line (method: “getClosestIntersection”). This is caused by the fact that the “Projectile” and “Line” class use the same visualization (a straight line) and thus the calculation of collision happens in the same way.

b.2) ***Fix the design flaw or extensively and precisely explain why this detected flaw is not an error and, thus, should not be fixed (10 pts)***
To fix this issue we have to get the intersection point calculation of a line outside of the “Line” and “Projectile” class. This calculation is very specific for lines, so we decided to create an “AbstractLine” class and extend this class with the “Line” and “Projectile” class. In the “AbstractLine” class we have put everything which belongs to a line object, except for the lineview object, which is used to visualize the line. This allows child classes to have different kinds of visualizations of the line (for example different colors, widths, opacity, etc.).
These changes have successfully fixes the detected design flaw. We have verified this by running inCode again: the sibling duplication flaw is fixed and there are no new flaws detected because of this change.

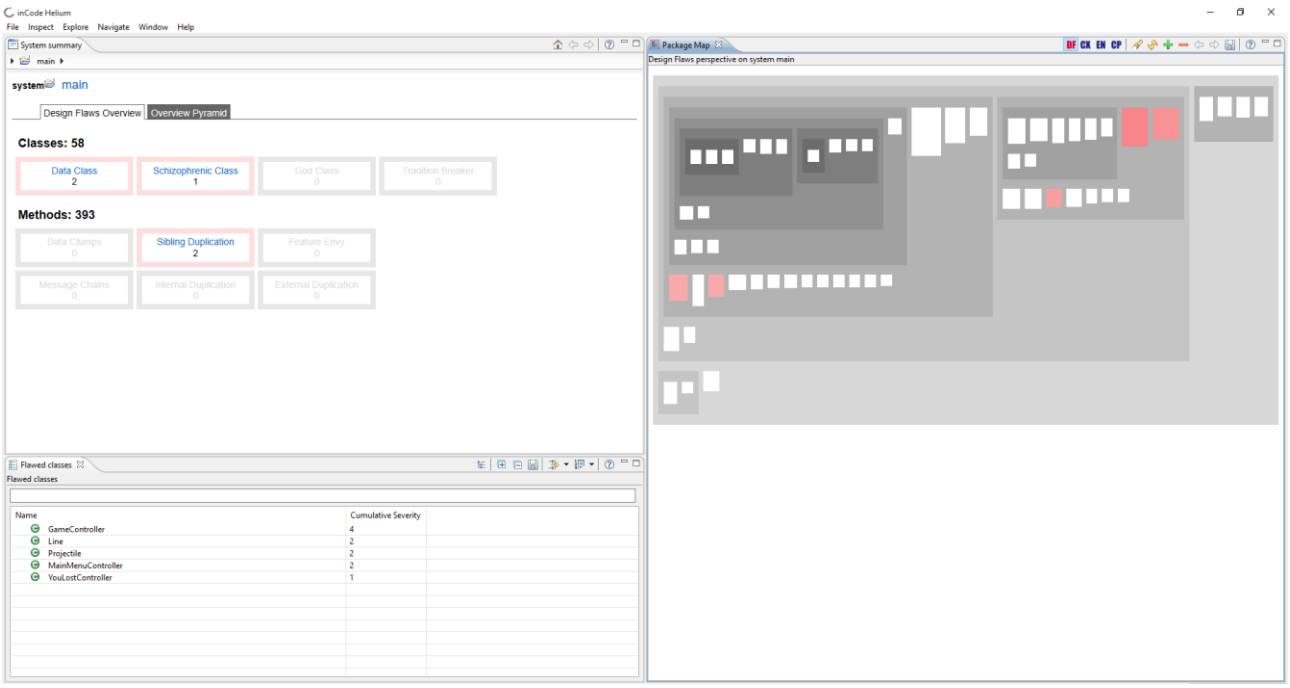
a.3) ***Explain the design choices or errors leading to the detected design flaw (4pts)***
Data class: “MainMenuController” and “YouLostController”
With the data classes there are 2 issues:
1) A lot of the variables of the classes are exposed through public methods (public getters).
2) There seem to be many unused methods.
Both of the ‘flaws’ detected by inCode are incorrect.
All of the private methods which aren’t called by the class itself got an @FXML tag and are linked to buttons in the fxml file (see exercise ‘b.1’ for an explanation of fxml files). This means that when you click on a button, the private methods get called. All of this is handled by javaFX, so it can’t be detected by inCode.
All variables which are exposed through public getters are only intended for testing purposes. They are not used in any of the non-test classes. These getters were necessary to create the integration tests: we had to fire button events to switch screens, thus we needed public getters for all of the buttons.

b.3) ***Fix the design flaw or extensively and precisely explain why this detected flaw is not an error and, thus, should not be fixed (10 pts)***
As described in exercise ‘a.3’, the detected ’flaws’ aren’t actually flaws. The private methods which seem to be unused are actually used by the fxml file objects and the public getters are used only for testing purposes.
To make the design a little bit better we’ve changed the public getters to protected getters, as test classes got the same package hierarchy. The classes are still detected as data classes by inCode after this change, but at least the variables are encapsulated a bit better.
Because the buttons which have been accessed through public getters were actually only used to fire events on those buttons, we have changed the getters to methods which directly fire the event of the button (so no public access is given to the buttons themselves). This actually fixed the inCode ‘flaw’, while a few public getters for other view elements are still present. Even though the ‘flaw’ isn’t visible in inCode anymore, we still treated it as a ‘flaw’ which we didn’t have to fix for this assignment, because it wasn’t a real flaw. The changes from public getters to methods which fire buttons is however a small design improvement, so we will keep this change in the code.

For convenience the analysis file after the code changes has also been put in the same folder as this pdf. It is called: “main_1444935805358.result”.

In case any problems occur when trying to import the .result files with inCode two images of the before/after view have been placed below (you have to place the snapshots in the snapshot folder of the inCode application for this to be possible, if you rename the snapshot file inCode seems to fail to detect it).

BEFORE:



AFTER: