

1: Metrics

Method based metrics

method	ev(G)	iv(G) ▼	v(G)
grid.Grid.stickInGrid(CollisionSet)	4	3	20
grid.Grid.equals(Object)	9	8	13
grid.Grid.collision(ShotBubble,int,int)	7	9	13
grid.Grid.getNeighbours(Coordinate)	1	12	12
game.Game.mainLoop()	1	11	11
gui.GameController.update(UpdateType)	2	2	9
grid.Grid.populateGrid(Scanner,Grid,int,int)	5	5	9
grid.Grid.getConnected(int,int,boolean)	3	7	8
grid.Grid.calculateSide(ShotBubble,Coordi	6	1	8
grid.CollisionSet.equals(Object)	3	6	8
grid.bubble.ShotBubble.equals(Object)	3	6	8
gui.MainScreenController.handlePlayButto	1	7	7
gui.GameController.updateGrid()	5	5	6
grid.bubble.GridBubble.equals(Object)	3	4	6
grid.Grid.putBubble(Bubble,boolean)	4	4	6
auth.Player.equals(Object)	3	4	6
grid.Coordinate.equals(Object)	3	2	5
grid.bubble.ShotBubble.move(int,int)	1	5	5
gui.StartController.handleLoginButtonAct	5	4	5

For the method specific metrics we used MetricsReloaded to look at the different types of complexities for the methods. Here ev(g) is essential complexity, iv(g) is Module-design complexity and v(g) is Cyclomatic complexity.

MetricsReloaded already highlights some methods which it thinks are problematic. From these we decided that we pick the top five bad Cyclomatic Complexity (CC) methods and want to get their CC below 10. However, for the collision method this seemed very complex because the method is inherently complex without a clear way to split it. Same with populateGrid. Therefore we decide to also improve calculateSide, since it has an essential Complexity (EC) of 6, and we would like this be 5 or lower for all methods, and improving the CC for the other methods should also already improve the EC and a CC of 8 which is still very high.

Class based metrics

Analysis of template

General Information

Total lines of code: 1376

Number of classes: 25

Number of packages: 5

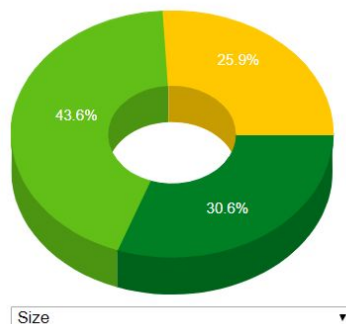
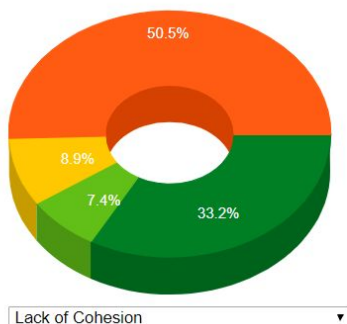
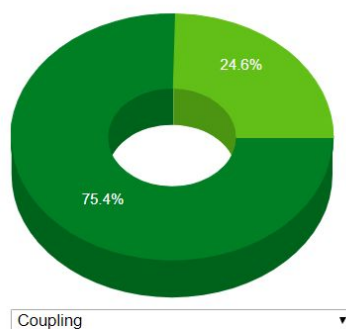
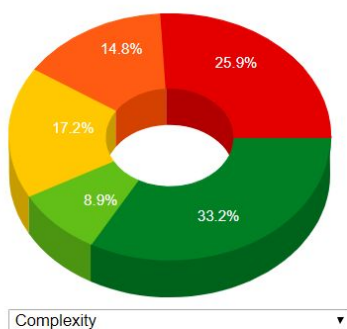
Number of external packages: 27





































Number of external classes: 96

Number of problematic classes: 2

Number of highly problematic classes: 0

The metric which represents the project as a whole:



List of all classes (#28)										
ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	GameController					227	high	low-medium	high	low-medium
2	Game					150	medium-high	low-medium	high	low-medium
3	Grid					396	very-high	low	high	medium-high
4	Main					54	medium-high	low	low	low-medium
5	MainScreenController					38	medium-high	low	low	low
6	WinController					22	medium-high	low	low	low
7	ShotBubble					89	low-medium	low	medium-high	low-medium
8	GridBubbleFactory					21	low-medium	low	low	low
9	ShotBubbleFactory					19	low-medium	low	low	low

For class based metrics we used CodeMR to determine the metrics. This does abstract some of the exact data away, because of that we are currently working with the info CodeMR provides. It informs us about the metrics of classes with very-high to low. Where high is bad and low is good.

Not unexpectedly the grid class pops up as a big problem child. It contains a lot of functionality, some of which does not belong in the class or would maybe be better fit in another class. Because of this it has very high complexity as a class and the cohesion could also be improved. The game class and Main class also pop out as problematic tasks. Ideally we'd also refactor the GameController class, but are currently unable to do so.

2. Refactoring

Method level Metrics

grid.stickInGrid()

This method had a Cyclomatic complexity of 20, that's a big yikes. By splitting up the functionality over different methods and making it more manageable we got the CC down to 6, where the two added methods both have a CC of 8.

Before:

method	ev(G)	iv(G) ▼	v(G)
grid.Grid.stickInGrid(CollisionSet)	4	3	20

After:

grid.Grid.stickInGrid(CollisionSet)	1	2	6
-------------------------------------	---	---	---

grid.equals()

This method seems necessary to be complex at times, however by adding some simplifications we managed to lower the CC from 13 to 9, which we consider barely acceptable.

We removed the null check since the class check already takes care of that. We also improved the part that checks if all the bubbles are the same by using Arrays.equals on each column. The EC metric here is still “red” as determined by MetricsReloaded but because of the nature of the equals statement we do not see a good solution for this currently and consider 5 EC good enough.

Before:

method	ev(G)	iv(G) ▼	v(G)
grid.Grid.equals(Object)	9	8	13

After:

grid.Grid.equals(Object)	5	6	9
--------------------------	---	---	---

grid.getNeighbours()

The reason this method is so complex is because of our grid representation. This means there is mainly an odd row and even row case. Because of that we split those two options into separate methods. Which makes the code more understandable and logically makes

sense. With this we brought the CC down from 12 to 4, where the two new methods both have a CC of 4 as well.

Before:

method	ev(G)	iv(G) ▼	v(G)
grid.Grid.getNeighbours(Coordinate)	1	12	12

After:

grid.Grid.getNeighbours(Coordinate)	1	4	4
-------------------------------------	---	---	---

grid.calculateSide()

method	ev(G)	iv(G) ▼	v(G)
grid.Grid.calculateSide(ShotBubble, Coordi	6	1	8

Similar to getting the neighbors, there is a bunch of logic involved into calculating the side of a bubble based on collision. However, the main cases are left side and right side. Because of this the logical choice was to also split parts of the method into these. We took the essential complexity from 6 down to 2, with both new methods having an essential complexity of 3. This also lowered the CC from 8 to 2, which is nice but was not a main requirement.

grid.Grid.calculateSide(ShotBubble, Coordinate)	2	2	2
---	---	---	---

game.mainLoop()

For the main game loop a lot of things have to occur, however by splitting it into sizeable chunks, where the parts that only occur when a shot bubble is currently moving, are in a separate methods we improve the CC from 11 into two methods of 6.

Before:

method	ev(G)	iv(G) ▼	v(G)
game.Game.mainLoop()	1	11	11

After:

game.Game.mainLoop()	1	6	6
game.Game.firedLoop()	1	6	6



Class level Metrics

Grid class

There is a lot of logic in the Grid class. The collision, determining of where the bubble in the real world is and all of the logic for creating a grid from a file is in here. This makes the class very complex and not coupled enough. We moved the collision logic to a new class called Collision. The creating a grid from a file is moved to a new class called GridCreator. Because of this both the complexity and coupling have gotten a lot better.

Before:

List of all classes (#30)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
3	Grid					396	very-high	low	high	medium-high

After:

List of all classes (#30)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
4	Grid					212	medium-high	low	medium-high	medium-high

The complexity has gone down from very-high to medium-high. The lack of cohesion has gone down from high to medium-high.

We were not able to get it even further down, because of the nature of the grid class.

Main class

The Main class only had a medium-high complexity. We chose to improve the main because it was in the top 5 worst classes.

To improve the main, the logic for loading a scene is moved to a new class called SceneLoader. This class now takes care of the loading of the scene. This caused the complexity of the main to go down to low-medium. The complexity of SceneLoader is medium-high, however, we feel like this is an improvement because of the nature of the Main class. The complexity is also improved, because the SceneLoader only has one purpose. This is loading scenes. This causes it to be understandable.

Before:

List of all classes (#30)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
4	Main					54	medium-high	low	low	low-medium

After:

List of all classes (#30)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
10	Main					9	low-medium	low	low	low

Game class



The game class had a high lack of cohesion. This is because the game class had the centerGrid method in it. The centerGrid method is moved to the Grid class, where it belongs more. This caused the lack of cohesion to now go down to medium-high. The complexity and lack of cohesion cannot be further lowered, because all of the logic that is in the Game class cannot be moved or made less complex.

Before:

2	Game					155	medium-high	low-medium	high	low-medium
---	------	---	---	---	---	-----	-------------	------------	------	------------

After:

List of all classes (#30)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
2	Game					131	medium-high	low-medium	medium-high	low-medium

Other classes

We did not refactor any other classes. This was because of a lack of time, and a lack of bad classes. As seen in the list of all classes and how they score, only the GameController is also a bad class. However, because of the nature of the GameController this is very difficult to refactor. This is because the GameController needs to call a lot of different methods. This causes the complexity to go drastically up. We were not able to refactor the GameController before the deadline. We were not able to refactor the GameController before the deadline.

Unconventional Design Decisions

As far as patterns go, in the end our factory pattern for the bubbles isn't quite as it should be. This is because there are some differences in the way GridBubbles and ShotBubbles behave, specifically requiring floats vs integers for position. Because of this both the grid and cannon have GridBubbleFactory and ShotBubbleFactory respectively, rather than the abstract BubbleFactory.

We see a potential solution in this by also creating a two different types of coordinates and a pattern to abstract that away as well, but we did not have the ability or time to accomplish this within the timeframe of the factory assignment or finishing of the project.

Another perhaps unconventional decision is the fact that the Main class, which technically is part of the UI, keeps track of the player. This choice has been made because the Main class is the connecting component between the Login Screen and the different screens. which makes it very useful for transferring player data between screens. Since not all screens have access to the game.