Assignment 4
Bubble Spinner Group 20

Contents

Contents **1**

1. Bubble Island **2**

1.1 The game 2

1.2 Reflecting on the requirements 5

1.3 Software properties 5

2. Analysis & refactoring **6**

2.1 Chidamber and Kemerer metrics 7

Weighted Methods for Class (WMC) 8

Coupling between objects (CBO) 8

Response for class (RFC) 9

Lack of cohesion of methods (LCOM) 10

Depth of inheritance three (DIT) 11

Number of children (NOC) 12

2.2 Refactoring 12

PlayScreen 12

GeneralScreen 12

CollisionHandler & CollisionHelper 13

Bubble class 13

BubbleBoard 14

Hud 14

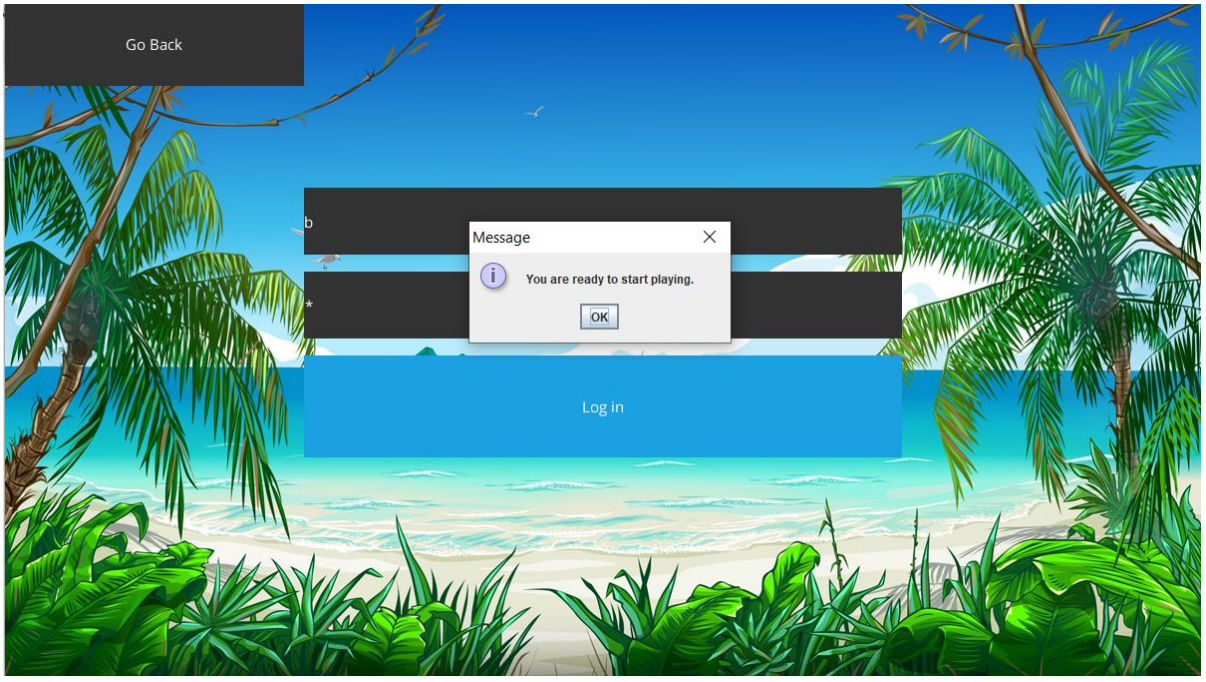
3. Contributions **15**

1. Bubble Island

1.1 The game

Our Bubble Spinner game is now fully functioning and has been given the name “Bubble Island”. It has a playful, user friendly GUI and offers a unique feature: because the bubbles contain different gemstones, they are distinguishable not only by color but also by shape. Therefore our product is also playable for colorblind users.

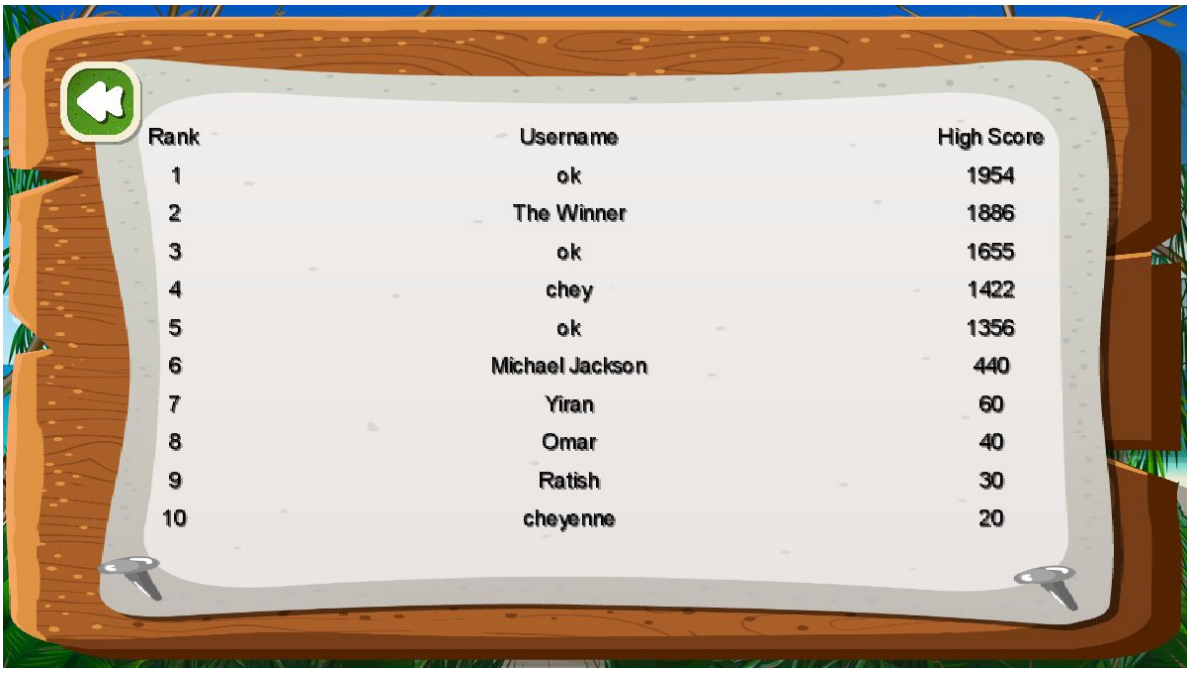
A player can create a new account or login to an already existing one.



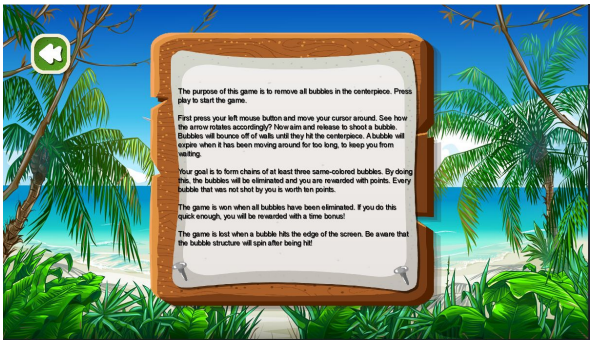
After having successfully authenticated, the user will be shown the start screen. From here he can choose to sign out, have a look at the leaderboard or start a new game.

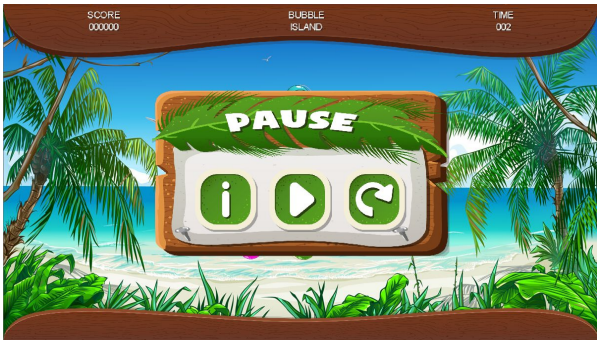
When a new game is started, a hexagon of bubbles is drawn. The player can eliminate them by forming chains of at least three same-colored bubbles. Bubbles that get detached from the centerpiece when other bubbles pop will also be removed. Both ways of popping bubbles will increase the score, which is shown on the screen together with a timer. The central structure will rotate when the player hits it by shooting a bubble. A player loses when this centerpiece hits and edge of the screen of collides with the bubble at the bottom of the screen (that is waiting to be shot). When all bubbles have been popped, the level is won and a time bonus is calculated and added to the score. The game either starts the next level, which has more bubbles to be popped, or ends when all levels are won.

Both winning and losing the game will let the user enter their name that will be saved in the database together with their score. Afterwards, the leaderboard is updated and the ten best scores are displayed.



A game manual and a pause function are also provided.





1.2 Reflecting on the requirements

All must-haves from the requirements document have successfully been implemented. In addition, multiple should-haves are implemented as well. Looking back at the requirements, some of them turned out to be infeasible for our team to implement within the time constraints. For example the badge system, where players are rewarded for certain achievements, should have been listed as a could-have requirement. This feature was not a necessity for our game to be playable or enjoyable. None of the can-have requirements have been implemented and, as expected, none of the won't-have requirements have been either.

The requirements that are not yet implemented can serve as guidelines for future versions of the game. Firstly, the reward system that was mentioned before could motivate users to keep on playing. Winning the first level or starting five new games are examples of such achievements. Secondly, a friendship system as described in the requirements would

be a valuable addition. At this point, user can already see scores of others on the leaderboard. This is because our system saves all scores in a database hosted on an external server. This database can be easily extended with additional tables to keep track of friendships and achievements. Players could then see the badges that their friends have earned, in addition to only comparing scores. A third extension would be to give users the ability to change their credentials and delete their account.

1.3 Software properties

Our development team has an abundance of ideas on how to extend the game and make it even better. Therefore we made sure to build our code open to extension and close to modification. Using the “State” design pattern, we gave the bubbles a static or dynamic state. This way the behaviour of bubbles can change, making it look as if they are different instances of classes. The pattern avoids large conditional if-statements that would arise if more states were to be added. Other states could cause the movement of bubbles to be randomized or cause bubbles to be gravitated towards the center (like bubbles shot by the game in a higher level). The usage of the “Iterator” pattern allows our play screen to update and render the bubbles in the board, without having knowledge about our having direct access to the internal implementation of the board. It also allows us to replace the board with different kinds (for example a triangle instead of a hexagon) in other levels.

The overall architecture adheres to model-view-controller. The player sees screens with icons and text, which correspond to the view. User input is processed in the controller part of the architecture by classes that take care of logic. An example is the userDao that handles authentication requests. It communicates with the database, which corresponds the the model. The databaseConnection class communicates back to the UserDao by means of booleans. Those are used by the UserDao to call the right methods of the UserDaoController, which creates pop-up messages that serve as a visual aid to give our users feedback. Therefore, this last class also belongs to the view. Using the model-view-controller architecture allows us separate logic from GUI, which makes testing easier. Units tests serve for logic classes, where GUI parts are tested manually. Extracting external dependencies, such as the database connection, allows for mocking the data such dependencies return.

2. Analysis & refactoring

We used the CodeMR plugin for IntelliJ to analyze our code. Below are a few graphs taken from the initial report. The two most problematic classes are PlayScreen, CollisionHandler and Bubble.

Legenda:

C3 (Coupling, Cohesion, Complexity)

Scope: Class

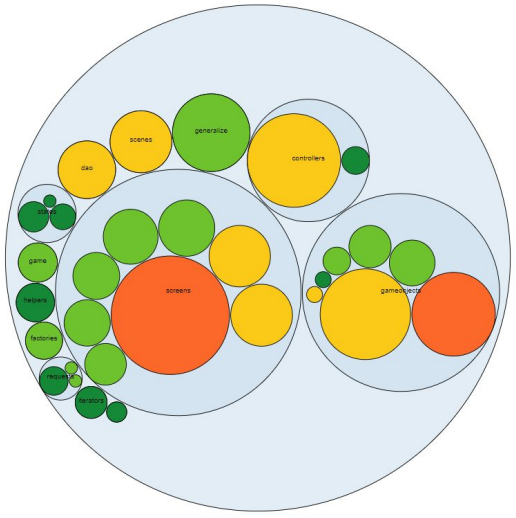
Description: The max value of Coupling, Cohesion, Complexity metrics

Related Quality Attributes: Coupling, Cohesion, Complexity

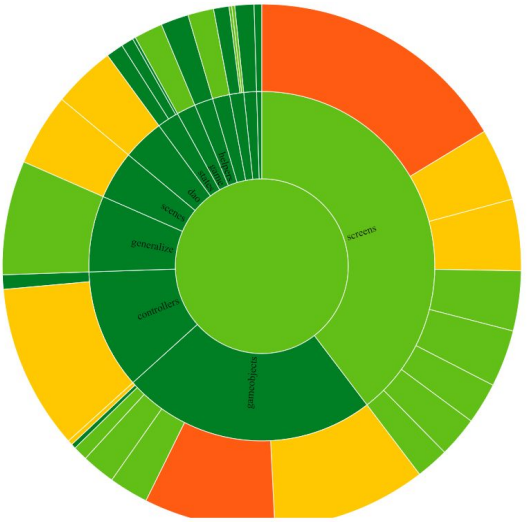
Click for more information

- Very high
- High
- Medium-high
- Low-medium
- Low

Metric values by package:



Metrics values in sunburst chart:



2.1 Chidamber and Kemerer metrics

We analysed our code through means of the CK metrics. This suite exists of six software metrics for object-oriented programs.[1]

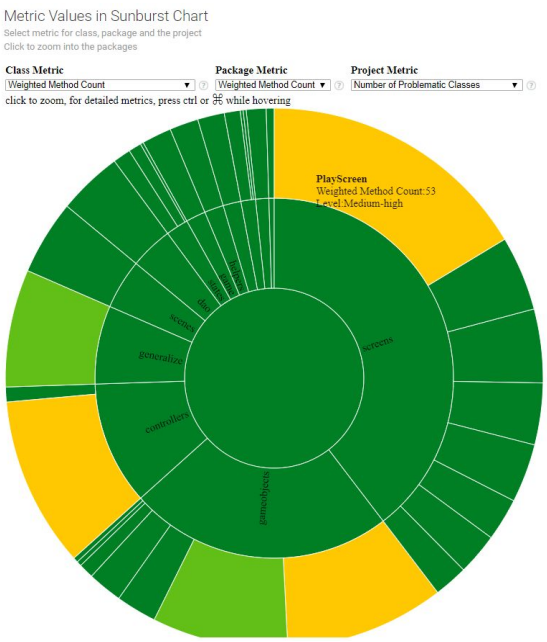
Weighted Methods for Class (WMC)

The graph below shows only three classes with medium to high problems. The following values for WMC were computed:

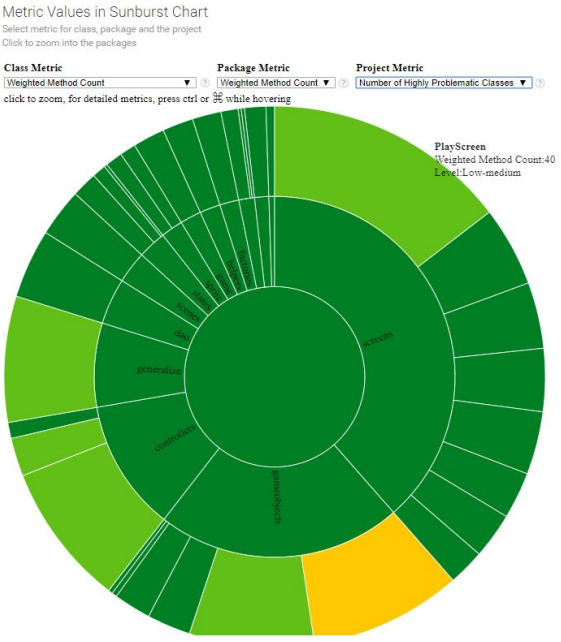
- PlayScreen: 53 (medium-high)
- CollisionHandler: 65 (medium-high)
- BubbleBoard: 66 (medium-high)

To target this issue, new classes can be extracted and methods can be moved.

Before:



After:



[1]S. Chidamber and C. Kemerer “A Metric Suite for Object Oriented Design” IEEE Transactions on Software Engineering (TSE), 199

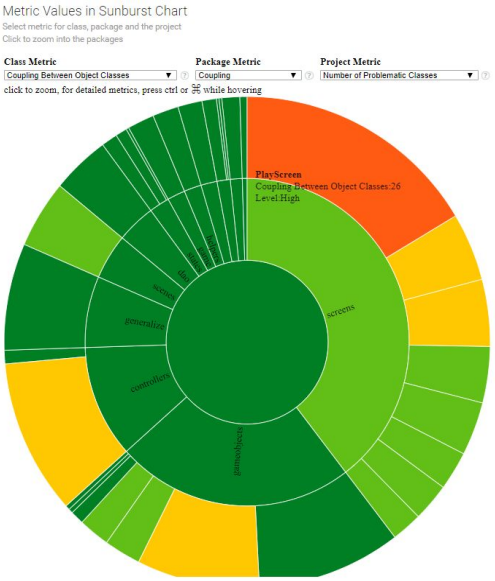
Coupling between objects (CBO)

The graph below shows fours classes with medium to high problems and one class with a high problematic score. The following values for CBO were computed:

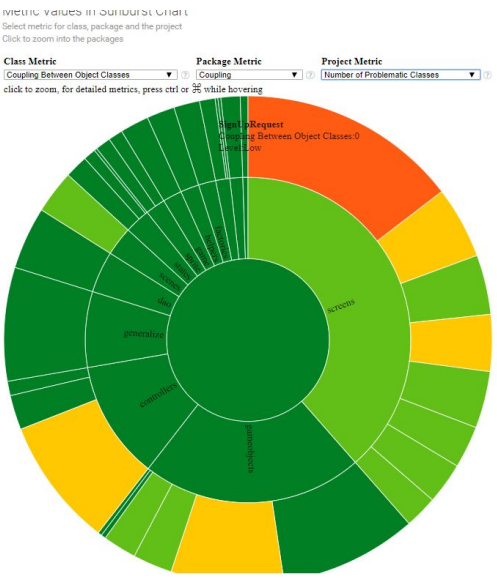
- PlayScreen: 26 (high)
- GeneralScreen: 12 (medium-high)
- LeaderBoardScreen: 13 (medium-high)
- Bubble: 12 (medium-high)
- CollisionHandler: 12 (medium-high)

Move method refactoring can be used to separate pieces and remove coupling. Classes can be extracted to factor out common interests. Delegation could replace inheritance.

Before:



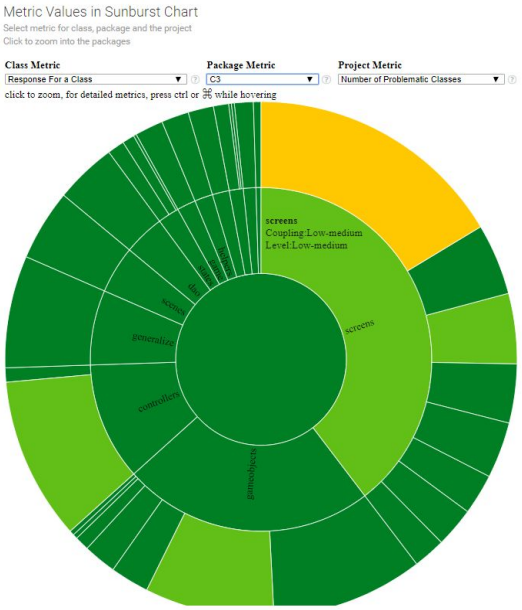
After:



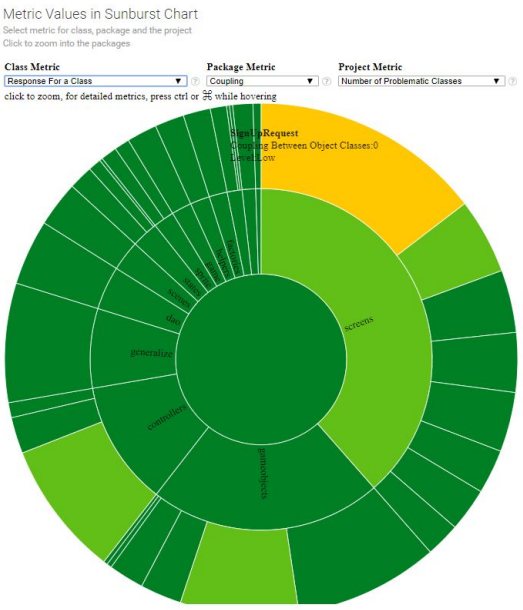
Response for class (RFC)

The response for class metric is the total number of methods that can potentially be executed in response to a message received by an object of a class.[2] There is one medium to high problematic class, namely PlayScreen with a score of 141.

Before:



After:



[2] https://www.oreilly.com/library/view/sonar-code-quality/9781849517867/ch09s04.html

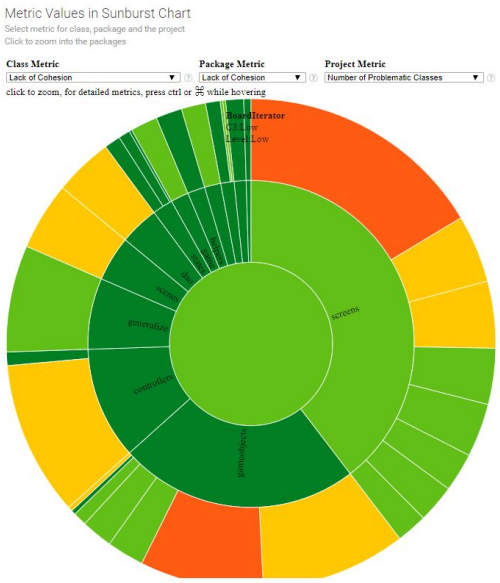
Lack of cohesion of methods (LCOM)

Some classes have more method pairs that do not share any attributes, than method pairs that share at least one attribute. This indicates lack of cohesion of methods within these classes. The most problematic classes are:

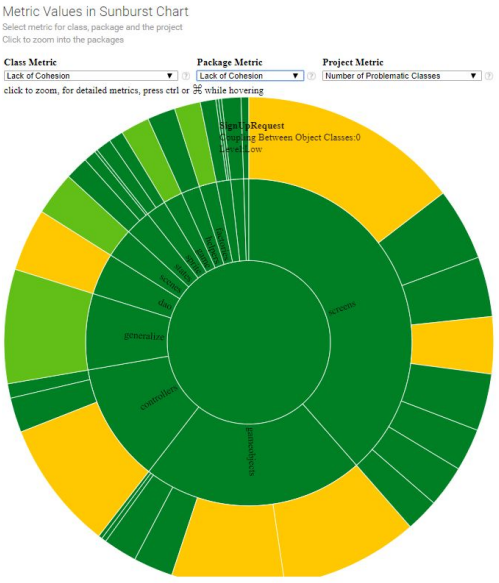
- PlayScreen: high
- GeneralScreen: medium-high
- LeaderBoardScreen: medium-high
- BubbleBoard: medium-high
- Bubble: high
- CollisionHandler: medium-high
- Hud: medium-high
- UserDao: medium-high

Classes can be extracted to split up responsibilities and increase cohesion.

Before:



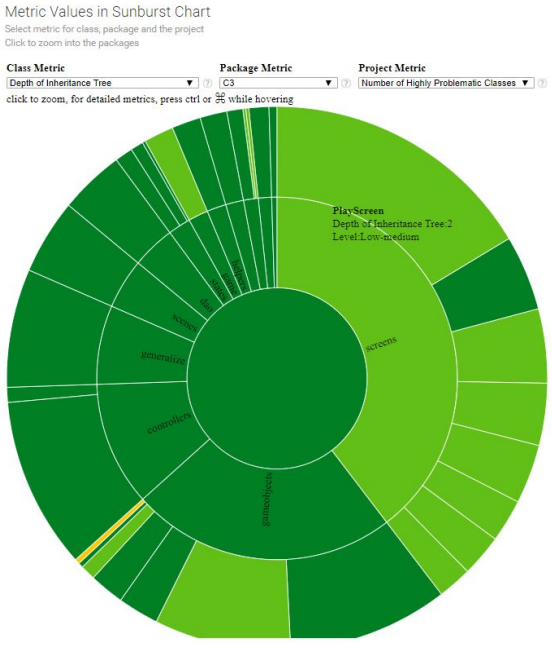
After:



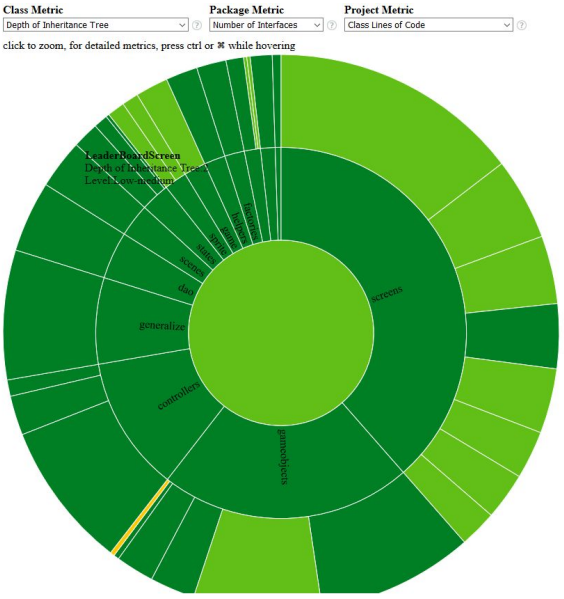
Depth of inheritance three (DIT)

Inheritance three's in our system are relatively simple. The screens all extend our abstract GeneralScreen. Some classes extend or implement libGDX classes. Only Star has medium-high problems, as its inheritance three has a depth of 4.

Before:



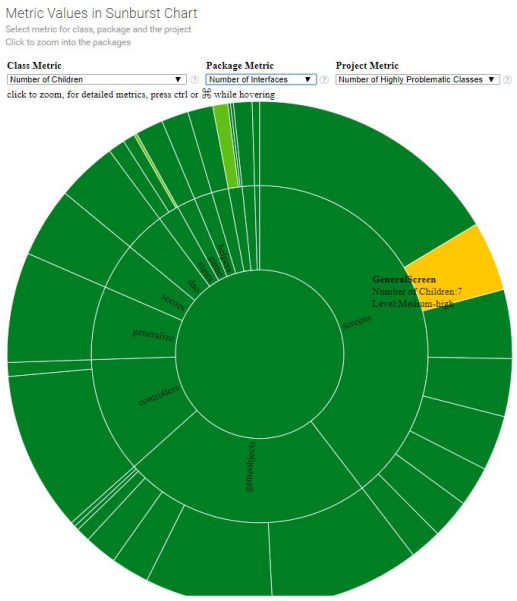
After:



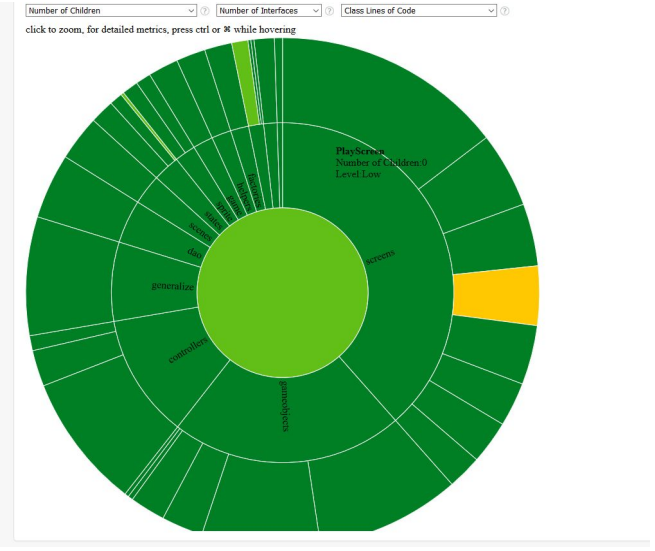
Number of children (NOC)

Most of our classes have a low number of children. Only the abstract GeneralScreen has seven children.This code smell does not indicate an issue. Previously, the screens all had a lot of duplicate code. This has been extracted to create the GeneralScreen class. This screen having seven children is better than having duplicate code in seven classes.

Before:



After:



2.2 Refactoring

PlayScreen

The Playscreen class has a high lack of cohesion, RFC score of 141 and a WMC score of 53. These all indicate an issue. In order to improve the cohesion, unnecessary methods that only use 1 or a few attributes like private getters and setters. We also removed attributes that did not have to be attributes like border, since it never gets used. Attributes and methods that should have been part of a different class were relocated to that class. This was for instance the case for the attribute bubble age and the method rotate. The lack of cohesion went from high to medium high, the RFC score went from 141 to 128 and the WMC score went from 53 to 40.

The coupling of the Playscreen constructor was very high, it has been reduced to medium high by removing unnecessary calls to other methods. Variables that could have been created in a more efficient way have also been changed.

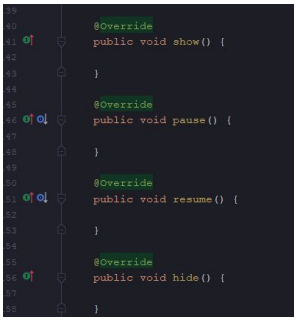
GeneralScreen

The GeneralScreen class has a medium-high CBO score of 12. It also has a medium-high LCOM score. The number of children score does indicate an issue, as is explained in the NOC paragraph of this report. The class does need better cohesion and lower coupling. The GeneralScreen contains a lot of file paths to button sprites. These are moved to the new enum class ScreenElements, in order to increase cohesion among methods for the screens. However, it is hard to get high cohesion among methods for the GeneralScreen, since it contains empty methods that override the methods of the Screen interface of libGDX. Those do not interact with class attributes or other methods. A lot of unnecessary getters and setters are deleted, which changed the class constructor and increased cohesion among class methods.

The deleted class attributes:

```
// File paths to images
static final String BACK = "assets/buttons/prev.png";
static final String BACKGROUND = "assets/background/island.jpg";
static final String GENERAL_SCREEN = "assets/displays/general.png";
static final String INFO = "assets/buttons/about.png";
static final String NAME = "assets/background/name.png";
static final String PAUSED = "assets/buttons/pause.png";
static final String PAUSED_SCREEN = "assets/displays/pause-screen.png";
static final String RANKING = "assets/buttons/ranking.png";
static final String RESTART = "assets/buttons/restart.png";
static final String PLAY = "assets/buttons/play.png";
```

The overridden methods:



CollisionHandler & CollisionHelper

We created a new helper class in order to reduce the complexity and coupling of the CollisionHandler class. The WMC has been reduced from high to low-medium. Since the collision handler is a controller class, it controls the behavior of the view and module classes. When some conditions are met, the controller calls methods from the other classes to alter their behavior.

A helper class can be used to reduce the coupling but that would cause a lack of cohesion between the main and helper class along with increasing the complexity.

All the functionality was handled in the beginContact method that is implemented by the contactListener interface. According to libGDX documentation this method should handle all the collisions in the game. However that method gained complexity very quickly.In order to reduce its complexity separate methods have been created. And a new helper class was created in order to communicate with the PlayScreen class further reducing the complexity of the class.

Bubble

Before refactoring:

Name	Coupling	Lack of Cohesion	WMC	CBO	RFC	LCOM	DIT	NOC
Bubble	medium-high	high	42	12	69	0.945	3	1

After refactoring:

Name	Coupling	Lack of Cohesion	CBO	RFC	DIT	NOC	WMC	LCOM
Bubble	medium-high	medium-high	12	59	3	1	32	0.917

The Bubble class used to have a high Lack of Cohesion with LCOM 0.945. We do not think extract class refactoring is appropriate in this case to reduce the lack of cohesion of Bubble class. Since all the methods of Bubble class are under the responsibilities the bubble object should have in the game. Making a separate helper class for Bubble class would make the data exchange process in the bubble object cumbersome and inefficient. Instead, we found there were some unnecessary methods included in the Bubble class, whose functions are largely overlapped with others. Therefore, we deleted the unnecessary functions, which in return gives us a better Lack of Cohesion result with LCOM 0.917.

BubbleBoard

Before refactoring:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	LOC	MCC
checkNeighbors(Bubble, ArrayList, int, int, Color): ArrayList	very-high	low	medium-high	low	2	63	39

After refactoring:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	LOC	MCC
checkNeighbors(Bubble, ArrayList, int, int): ArrayList	low-medium	low	low	low	1	27	15
addBubbleIntoComponent(Color, Bubble, ArrayList): void	low	low	low	low	2	8	5

The findNeighbors() method in the BubbleBoard class used to have a very-high complexity with which MCC equals 39. With extract method refactoring, the same code fragments in the findNeighbors() were grouped together and move to a new method called addBubbleIntoComponent(). As a result, the complexity of the findNeighbors() method is reduced to low-medium and with MCC 15. And the addBubbleIntoComponent() method also has a low complexity with MCC 5.

Hud

The Hud had a medium high lack of cohesion. The cohesion got improved by removing redundant variables and methods. The lack of cohesion went from medium high to low

medium, the CBO score went from 10 to 8, the RFC went from 31 to 20 and the WMC score went from 20 to 8.

3. Contributions

Listed below are the contributions during the final week, which fall outside of the final sprint and are therefore not included in any retrospective.

Name	Contribution	Time spend (h)
Cheyenne Slager	-Issue #97: Starfish -Assignment 4, 1. Bubble Island -Assignment 4, 2.1 Analysis -Assignment 4, 2.2: GeneralScreen	6 1 3 4
Omar Thabet	-Assignment 4, 2.2: CollisionHandler & CollisionHelper	4
Ratish Thakoersingh	-Assignment 4, 2.2: PlayScreen -Assignment 4, 2.2: Hud	6 2
Valencio Faerber	-Assignment 4, 2.2:	
Yiran Wang	-Issue #52: Bubble detachment -Assignment 4, 2.2: Bubble class -Assignment 4, 2.2: BubbleBoard	12 0.5 2