Refactoring assignment #4

We are using the plugin codeMR for Intellij to measure the metrics of our codebase. In particular, we are interested in reducing the coupling and the complexity of the classes, as well as to increase cohesion if possible. The thresholds to determine whether a class needs refactoring we use the one given by default from the codeMR plugin. For example, in the case of CBO (Coupling Between Objects), we refactor all classes with a threshold higher than 20 which shows as high/very-high in the report.

For the method level metrics we are using SourceMonitor windows tool that checks the complexity and depth of calls for each method present.. We decided to refactor methods that have a threshold greater than or equal to 10 for any of the following metrics: complexity, statements, max depth, calls.

Classes: RegisterScreen, MenuScreen, LoginScreen

After running the CodeMR plugin on the game package of the project the report stated that 3 classes had high coupling and therefore they needed to be refactored. The refactor commit can be found on gitlab at: 5491e0506100c84efb8c31a0d1038ae9d67ff2a9

All the buttons, labels, and fields found in the screens were moved to a separate class. For each screen there is a respective menu (MainMenu, RegisterMenu, LoginMenu) which handles the creation of the actors and their listeners. In order to make it easier to create UI elements now there is a UserInterfaceFactory class which stores the skin and the stylename. It contains 3 methods (createTextButton, createTextField, createLabel) to create the respective elements. The creation of the table contained a lot of duplicated code therefore it has been moved to a new class TableMenu which creates and adds rows automatically.

Before:

Classes with high coupling (#3)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	CBO	CBO APP	CBO LIB	RFC
1	MenuScreen					173	25	8	17	51
2	RegisterScreen					111	24	7	17	48
3	LoginScreen					106	21	4	17	43

2	MenuScreen					173	low-medium	high	low-medium	low-medium
3	RegisterScreen					111	low-medium	high	low-medium	low-medium
4	LoginScreen					106	low-medium	high	medium-high	low-medium

After:

Classes with high coupling (#0)

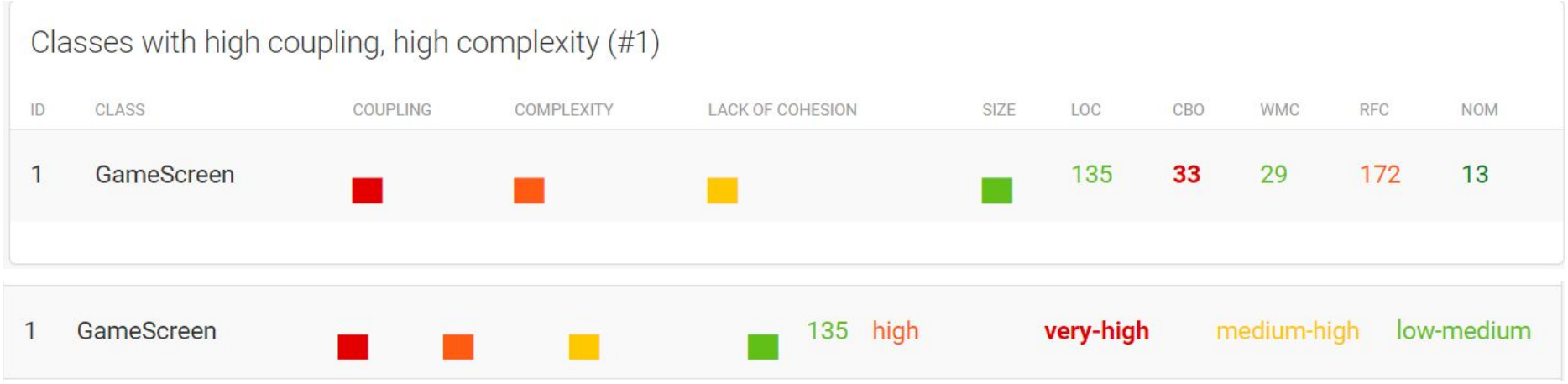
ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	CBO	CBO APP	CBO LIB	RFC
1	MenuScreen					65	low-medium	medium-high	medium-high	low-medium
2	RegisterScreen					55	low-medium	medium-high	low-medium	low-medium
3	LoginScreen					52	low-medium	medium-high	medium-high	low-medium
4	RegisterMenu					43	low	low-medium	low	low
5	LoginMenu					42	low	low-medium	low	low
6	TableMenu					17	medium-high	low	low	low
7	UIFactory					19	low	low	low	low
8	LabelColor					0	low	low	low	low

Class: GameScreen

The class GameScreen showed high coupling and complexity on codeMR. First refactored commit can be found at: 306e11f11540598e8917ccd04f0deb86005a5842

In order to reduce the coupling we extracted two classes from it. The part that draws the arrow is taken to its own class namely AimingArrow. We also introduced a new class, called GameMenu, which controls the pausing of the game and the displaying of the tutorial box.

Before:



After:

Class : DbImplement

The only refactored commit has SHA : 971832a89ed8cf1b7f57fb7b0d59b58b5aeaeda1

In order to reduce the dbImplement class complexity we needed to extract part of the class so that the class can have less tasks to do.Thats why dbImplementGet was created in order to handle getting the objects from the database.

Before :

After :

List of all classes (#10)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	DbImplement					137	low-medium	low-medium	medium-high	low-medium
2	DbImplementGet					114	low-medium	low-medium	low	low-medium
3	Server					77	low-medium	low-medium	medium-high	low-medium

Class: HexagonController

The final refactoring commit of hexagonController can be found at SHA: 71bce8f96ad8abb7cd53d8afa8668ebbd73a6284
The hexagonController class was reported as having medium-high complexity, therefore we worked on reducing that aspect first. One of the reasons the complexity was this high, could be attributed to the code responsible for setting up the hexagon. Namely, the hexagon changes its construction based on the difficulty of the level and we had all variations of constructions in a single method.

The way we fixed the complexity level was by applying the Strategy Design Pattern. As a solution we created three more classes pertaining to different ways of setting up the hexagon, all implementing the hexagonStrategy interface.

Before:

11	HexagonController					152	medium-high	low-medium	medium-high	low-medium

After:

15	HexagonController					124	low-medium	low-medium	medium-high	low-medium
33	HardHexagonStrategy					12	low	low	low	low
34	MediumHexagonStra...					10	low	low	low	low
35	EasyHexagonStrategy					10	low	low	low	low

Most Complex Methods in 1 Class(es):	Complexity, Statements, Max Depth, Calls
HexagonController.bubbleMissed()	2, 5, 3, 0
HexagonController.checkCollisions()	4, 14, 4, 15
HexagonController.checkGameStatus()	3, 4, 3, 4
HexagonController.drawGrid()	2, 6, 3, 6
HexagonController.formula()	4, 7, 3, 1
HexagonController.getBubbles()	1, 1, 2, 0
HexagonController.getMapBubbles()	1, 1, 2, 0
HexagonController.getPoppable()	4, 8, 4, 8
HexagonController.getResult()	1, 1, 2, 0
HexagonController.HexagonController()	1, 5, 2, 3
HexagonController.initialize()	42, 34, 6, 37
HexagonController.popBubbles()	3, 10, 4, 8
HexagonController.popFloatingBubbles()	3, 9, 4, 6
HexagonController.popSingleBubble()	1, 4, 2, 4

Method: BubbleSpinnerController.update()

The update method of the class includes a large amount of calls to different methods as well as statements. As a result, we decided to split it into multiple methods. We implemented two methods from this class to checkCurrentBubble and checkGameStatus, respectively, by extracting some code of the update() method and separating it into logical wholes.. This improved the cyclomatic complexity of the method and made the

class more testable. The commits can be found at 316fa90320d3d2c917ac8dd95f3b4f1ec589d906 and 323d75d138ac00b36e4b53f7ad6b94ba04d4218c

Before:

Most Complex Methods in 1 Class(es):	Complexity, Statements, Max Depth, Calls
BubbleSpinnerController.BubbleSpinnerController()	1, 4, 2, 2
BubbleSpinnerController.checkShoot()	4, 2, 3, 3
BubbleSpinnerController.getResult()	1, 1, 2, 1
BubbleSpinnerController.initialize()	1, 3, 2, 3
BubbleSpinnerController.setHexagonController()	1, 1, 2, 0
BubbleSpinnerController.setShooter()	1, 1, 2, 0
BubbleSpinnerController.update()	8, 14, 4, 16

After:

Most Complex Methods in 1 Class(es):	Complexity, Statements, Max Depth, Calls
BubbleSpinnerController.BubbleSpinnerController()	1, 4, 2, 2
BubbleSpinnerController.checkCurrentBubble()	6, 6, 4, 8
BubbleSpinnerController.checkShoot()	4, 2, 3, 3
BubbleSpinnerController.getResult()	1, 1, 2, 1
BubbleSpinnerController.initialize()	1, 3, 2, 3
BubbleSpinnerController.setHexagonController()	1, 1, 2, 0
BubbleSpinnerController.setShooter()	1, 1, 2, 0
BubbleSpinnerController.update()	1, 6, 2, 6

Method: Client.getUserBadge()

The method to retrieve and parse the user badges on the client side contained many calls and statements as well. In response, we decided to refactor it by extracting and dividing part of the original method into two methods parseBadges() and getImageFromBadgeName() which lessen the amount of work Client.getUserBadge() does. The commits can be found: e9151086080c5bcf0f6ec0c5c7cfe5f56f23bfc8 and 710ee94d52c015bdc46d7a4dd4b72b53bb02f58d

Before:

Most Complex Methods in 1 Class(es):	Complexity, Statements, Max Depth, Calls
Client.addBadge()	1, 3, 2, 2
Client.addScore()	1, 3, 2, 2
Client.addUser()	1, 3, 2, 2
Client.authenticate()	1, 3, 2, 4
Client.getBadges()	1, 1, 2, 2
Client.getSCore()	1, 1, 2, 2
Client.getTop5()	1, 4, 2, 3
Client.getUserBadge()	2, 10, 3, 17
Client.register()	1, 3, 2, 2

After:

Most Complex Methods in 1 Class(es):	Complexity, Statements, Max Depth, Calls
Client.addBadge()	1, 3, 2, 2
Client.addScore()	1, 3, 2, 2
Client.addUser()	1, 3, 2, 2
Client.authenticate()	1, 3, 2, 4
Client.getBadges()	1, 1, 2, 2
Client.getImageFromBadgeName()	1, 1, 2, 3
Client.getSCore()	1, 1, 2, 2
Client.getTop5()	1, 4, 2, 3
Client.getUserBadge()	1, 4, 2, 4
Client.parseBadges()	2, 5, 3, 7
Client.register()	1, 3, 2, 2

Method: BubbleFactory.createBubbleGivenMap()

The method to createBubbleGivenMap had the same problem as the previous two methods: too many statements and calls. We therefore extracted the code into a new method getPossibleColors which will retrieve a list of available colors from the hash map. The commit con be found at 8b59ffeb3951ba04c38e05e79fe4f47974ac42b8

Before:

Most Complex Methods in 1 Class(es):	Complexity, Statements, Max Depth, Calls
BubbleFactory.addAllTextures()	3, 3, 3, 1
BubbleFactory.addTexture()	1, 1, 2, 2
BubbleFactory.addTexture()	1, 1, 2, 1
BubbleFactory.BubbleFactory()	1, 1, 2, 1
BubbleFactory.BubbleFactory()	1, 3, 2, 1
BubbleFactory.createBubble()	1, 4, 2, 4
BubbleFactory.createBubbleGivenMap()	4, 11, 4, 10
BubbleFactory.createCenterBubble()	1, 4, 2, 3

After:

Most Complex Methods in 1 Class(es):	Complexity, Statements, Max Depth, Calls
BubbleFactory.addAllTextures()	3, 3, 3, 1
BubbleFactory.addTexture()	1, 1, 2, 2
BubbleFactory.addTexture()	1, 1, 2, 1
BubbleFactory.BubbleFactory()	1, 1, 2, 1
BubbleFactory.BubbleFactory()	1, 3, 2, 1
BubbleFactory.createBubble()	1, 4, 2, 4
BubbleFactory.createBubbleGivenMap()	3, 8, 3, 8
BubbleFactory.createCenterBubble()	1, 4, 2, 3
BubbleFactory.getPossibleColors()	3, 5, 4, 2

Method: HexagonController.initialize()

When we first wrote the initialize() method for the HexagonController, it was intended to take the load off of the HexagonController constructor's complexity. However, all of the complexity and high coupling index transferred to initialize() instead. In response, we extracted the less cohesive code into the BubbleSpinnerController class under the name difficultyLevel(). Another step was implementing the Strategy Design pattern to relieve the initialize() from the many if statements, specifically in the setUp().

Before:

Method: HexagonController.checkCollisions()

The method checkCollisions was very complex at first. Therefore we decided to split it up the part that was responsible for adding a bubble into the grid to another method called addBubbleToGrid(). This drastically decreased the complexity of the checkCollisions method.

Before:

Most Complex Methods in 1 Class(es):	Complexity, Statements, Max Depth, Calls
HexagonController.bubbleMissed()	2, 5, 3, 0
HexagonController.checkCollisions()	4, 14, 4, 15
HexagonController.checkGameStatus()	3, 4, 3, 4
HexagonController.drawGrid()	2, 6, 3, 6
HexagonController.formula()	4, 7, 3, 1
HexagonController.getBubbles()	1, 1, 2, 0
HexagonController.getMapBubbles()	1, 1, 2, 0
HexagonController.getPoppable()	4, 8, 4, 8
HexagonController.getResult()	1, 1, 2, 0
HexagonController.HexagonController()	1, 5, 2, 3
HexagonController.initialize()	42, 34, 6, 37
HexagonController.popBubbles()	3, 10, 4, 8
HexagonController.popFloatingBubbles()	3, 9, 4, 6
HexagonController.popSingleBubble()	1, 4, 2, 4

After:

Most Complex Methods in 1 Class(es):	Complexity, Statements, Max Depth, Calls
HexagonController.addBubbleToGrid()	1, 6, 2, 9
HexagonController.bubbleMissed()	2, 5, 3, 0
HexagonController.checkCollisions()	4, 9, 4, 7
HexagonController.checkGameStatus()	5, 9, 3, 3
HexagonController.drawGrid()	2, 6, 3, 6
HexagonController.formula()	4, 7, 3, 1
HexagonController.getBubbles()	1, 1, 2, 0
HexagonController.getBuilder()	1, 1, 2, 0
HexagonController.getMapBubbles()	1, 1, 2, 0
HexagonController.getPoppable()	4, 8, 4, 8
HexagonController.getResult()	1, 1, 2, 0
HexagonController.HexagonController()	1, 5, 2, 3
HexagonController.initialize()	1, 6, 2, 8
HexagonController.popBubbles()	3, 10, 4, 8
HexagonController.popFloatingBubbles()	3, 9, 4, 6
HexagonController.popSingleBubble()	1, 4, 2, 4
HexagonController.positionBubble()	1, 5, 2, 5
HexagonController.setBubbleFactory()	1, 1, 2, 0
HexagonController.setBuilder()	1, 1, 2, 0

Refactored Classes:

- GameScreen
- MenuScreen
- LoginScreen
- RegisterScreen
- DbImplement
- HexagonController

New Classes:

- GameMenu
- MainMenu
- LoginMenu
- RegisterMenu
- TableMenu
- AimingArrow
- UserInterfaceFactory
- DbImplementGet
- HexagonStrategy

Refactored Methods:

- BubbleSpinnerController.update()
- Client.getUserBadge()
- BubbleFactory.createBubbleGivenMap()
- HexagonController.initialize()
- HexagonController.checkCollisions()

New Methods:

- BubbleSpinnerController.checkCurrentBubble()

- HexagonController.checkGameStatus()
- HexagonController.positionBubble()
- Client.parseBadges()
- Client.getImageFromBadgeName()
- BubbleFactory.getPossibleColors()
- BubbleSpinnerController.difficultyLevel()
- HexagonController.addBubbleToGrid()
- HexagonStrategy.setUp()

List of missing requirements:

- Should-have R2 Dark-light mode
- Should-have R4 Special Bubbles
- Could-have R3 Friendlist
- Could-have R4 Password recovery
- Could-have R5 Multiplayer
- Could-have R7 Player vs Computer
- Could-have R8 Loading Screen