Tools used:

There are main tools that we used:

- CodeMR (for classes)
- Metrics reloaded (for methods)

Code smells analysis : Classes

We ran into a variety of issues when we tried to improve and refactor our classes. One issue that we had at the start of refactoring was that we depended on the Board class to generally to run the game. This resulted in the class being huge and having an issue with Complexity, Cohesion, and Coupling. To deal with this we decided to split up the Board class into one main overarching class for the game, this did result in a large class with High lack of cohesion and high coupling but this had to be expected since one class did have to be responsible with interconnecting components. These are the classes we decided to refactor:

- Board Class
- Game Class
- PowerUps Classes
- GameScreen/GameOverScreen/PauseMenu
- LoginScren/RegisterScreen

The metrics we used to identify classes that needed to be fixed

Thresholds:

- We used the inbuilt descriptions from the tools to measure what classes to focus on
- Method's containing a lack of cohesion was something we targeted for deciding what classes to refactor

The full report

Before:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT	NOC	WMC	LOC	CMLOC	NOF	NOSF	NOM
DesktopLauncher	low	low	low	low	3	1	0	1	0	1	7	5	0	1	0
com.snake.game.game	low	low-medium	medium-high	low						147	430				
Apple	low	low	low	low	4	21	12	1	0	10	32	29	2	0	6
Board	medium-high	medium-high	medium-high	high	11	102	32	1	0	57	162	144	9	8	36
DirectionQueue	low	low	low	low	1	14	6	1	0	8	18	16	1	0	6
Game	low	low	low	low-medium	5	22	4	1	0	10	26	22	3	0	10
Score	low	low	low	low	2	23	6	1	0	6	18	16	1	0	5
ScoreLabel	low	low-medium	low	low	7	9	7	1	0	2	22	13	4	4	2
Snake	low-medium	low	low-medium	low-medium	2	30	13	1	0	21	92	51	3	0	14
Timer	low	low	low	low-medium	1	12	1	1	0	15	36	31	4	0	11
User	low	low	low	low	0	9	1	1	0	9	24	20	2	1	7
com.snake.game.powerup	low	low-medium	low-medium	low						41	176				
LengthPowerUp	low-medium	low	low	low	5	13	10	2	0	4	17	13	0	3	3
MegaApple	low-medium	low	low	low	5	12	9	2	0	3	16	13	0	2	3
MoreApples	low-medium	low	low	low	5	22	9	2	0	3	16	13	0	2	3
PowerUp	low	low	low	low	2	11	0	1	5	11	28	23	4	0	11
PowerUpFactory	low	low-medium	low	low-medium	9	14	4	1	0	14	45	41	3	0	8
PowerUps	low-medium	low	low	low	0	0	0	2	0	0	7	0	0	6	0
SpeedUp	low-medium	low-medium	low	low	6	14	11	2	0	3	25	24	0	0	3
StopGrow	low-medium	low	low	low	5	13	10	2	0	3	22	20	0	1	3
com.snake.game.requests	low	low	low	low						35	124				
ApiRequest	low	low	low	low	3	11	3	1	4	8	19	15	2	1	8
Login	low-medium	low	low	low	5	14	10	2	0	7	28	25	2	0	2
MaxScore	low-medium	low	low	low	5	15	11	2	0	6	25	22	2	0	2
Signup	low-medium	low	low	low	5	14	10	2	0	8	32	28	2	1	2
UserInfo	low-medium	low	low	low	4	12	8	2	0	6	20	18	1	0	2
com.snake.game.screens	low	low-medium	low-medium	low						83	591				
GameOverScreen	low-medium	medium-high	low-medium	low-medium	15	29	17	2	0	14	90	82	7	0	14
GameScreen	low-medium	medium-high	low-medium	low-medium	20	90	25	2	0	23	80	72	7	0	12
LoginScreen	low-medium	medium-high	low-medium	low	14	25	18	2	0	7	103	93	9	0	7
PauseMenu	low-medium	medium-high	low-medium	low-medium	13	20	13	2	0	9	98	87	10	0	9
RegisterScreen	low-medium	medium-high	low-medium	low	14	24	16	2	0	6	77	68	8	0	6
Screen	low	low-medium	low-medium	medium-high	9	24	12	1	6	12	54	49	2	2	10
ScreenController	low-medium	low-medium	low	low	10	10	6	2	0	5	26	21	2	0	5
StartScreen	low-medium	medium-high	low-medium	low	13	17	12	2	0	7	63	55	7	0	7
com.snake.game.states	low	low	low	low						12	36				
ActiveGame	low	low	low	low	4	23	6	1	0	4	13	11	1	0	3
FinishedGame	low	low	low	low	1	12	2	1	0	3	10	8	1	0	3
PausedGame	low	low	low	low	2	20	3	1	0	3	10	8	1	0	3
State	low	low	low	low	0	2	0	1	3	2	3	2	0	0	2

After:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT
template								
com.snake.game	low	low	low	low				
DesktopLauncher	low	low	low	low	2	1	0	1
com.snake.game.game	low	low-medium	medium-high	low				
Apple	low	low-medium	low	low	6	26	18	1
Board	low	low-medium	low	low	6	24	10	1
Consumable	low	low	low	low	2	2	0	1
DirectionQueue	low	low	low	low	1	14	6	1
Game	low-medium	medium-high	medium-high	high	18	83	31	1
Score	low	low	low	low	2	24	7	1
ScoreLabel	low	low-medium	low	low	7	12	8	1
Settings	low	low	low	low	0	16	8	1
Snake	low-medium	low	low-medium	low-medium	3	33	15	1
SoundSystem	low	low	low	low	5	9	2	1
SoundWrapper	low	low	low	low	2	8	3	1
Timer	low	low	low	low-medium	1	13	2	1
User	low	low	low	low	0	9	1	1
com.snake.game.game.powerup	low	low-medium	low-medium	low				
LengthPowerUp	low-medium	low-medium	low	low	6	15	12	2
MegaApple	low-medium	low-medium	low	low	6	14	11	2
MoreApples	low-medium	low-medium	low	low	6	24	10	2
PowerUp	low	low	low	low	0	3	0	1
PowerUpFactory	low	low-medium	low	low	10	14	6	1
PowerUpName	low-medium	low	low	low	0	0	0	2
SpeedUp	low-medium	low-medium	low	low	7	16	12	2
StopGrow	low-medium	low-medium	low	low	6	14	11	2
com.snake.game.game.states	low	low	low-medium	low				
ActiveGameState	low-medium	low-medium	low	low	10	21	7	2
FinishedGameState	low-medium	low-medium	low	low	10	19	7	2
GameState	low	low	low	low	2	5	0	1
GameStateName	low-medium	low	low	low	0	0	0	2
NewGameState	low-medium	low-medium	low	low	10	33	9	2
PauseGameState	low-medium	low-medium	low	low	9	13	6	2
com.snake.game.requests	low	low-medium	low-medium	low				
ApiRequest	low	low	low	low	4	13	4	1
Leaderboard	low-medium	low	low	low	3	9	5	2
Login	low-medium	low	low	low	5	14	10	2
MaxScore	low-medium	low	low	low	5	15	11	2
Signup	low-medium	low	low	low	5	14	10	2
User	low	low	low	low	0	5	0	1
UserInfo	low-medium	low	low	low	4	12	8	2
com.snake.game.screens	low	low-medium	low-medium	low				
Font	low	low	low	low	4	4	3	1
GameScreen	low-medium	high	low-medium	medium-high	26	80	42	2
LeaderboardScreen	low-medium	medium-high	low-medium	low-medium	20	54	34	2
MenuScreen	low-medium	medium-high	low-medium	low-medium	20	43	33	2
Screen	low	low	low	low-medium	4	10	3	1
ScreenController	low-medium	low-medium	low	low	6	10	5	2
StartScreen	low-medium	medium-high	low-medium	low-medium	17	32	22	2

Code smells analysis : methods

High cyclomatic complexity appears to be the biggest method code smell we have in our game: small methods that take high responsibility is the first thing we have to fix. It is always easier to have smaller methods being responsible for smaller tasks because it's easier to track down the responsibility of each segment of a code.
Main methods that we have problems with are:

- Signup.execute()
- Login.execute()
- Apple.isProperSpawnLocation()
- Game.updateDirection(Snake.Direction direction)

There are only 4 methods which we had to refactor based on essential cyclomatic complexity with a threshold being 4.
The metrics we used to identify method smells are cyclomatic complexity, design complexity, cyclomatic complexity, long methods, long parameter list. In most of the cases there are almost no problems being present: we don't have any methods with too many parameters, or methods with too many lines of code. We have few methods with design complexity, but they belong to the GUI part of our code, so there isn't an option to refactor these methods.
Thresholds:

- Cyclomatic complexity - 9
- Essential cyclomatic complexity - 4
- Lines of code - 200
- Long parameter list - 6
-

9 Class-Level Code Smell Refactors

Decreasing Complexity in Board Class

The board class contained both all the sizes and metrics for the graphics, as well as the methods and fields for the game sprites, such as the `Snake`, `Apple` and `PowerUps`. To dampen this excessive size, we moved the game sprites to the `Game` class, which was considerably smaller despite its informal use as a “method/field hub”, meaning that almost all methods that act upon the game or its states would act upon the `Game` class. Also, methods needing the `Board` class would often get the `Game` passed as parameter and may use the `Game.getBoard()` getter. All this refactoring made it so that we moved all the main logic from the board class over to the game class. This completely removed most of the complexity and cohesion/coupling issues that are **Board** class had since we moved most of those issues over to **Game** Class.

As a result CodeMr Shows a significant improvements in the complexity/cohesion/coupling of the **Board** Class

Before:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT
template								
com.snake.game.game	low	low-medium	medium-high	low				
Apple	low	low	low	low	4	21	12	1
Board	medium-high	medium-high	medium-high	high	11	90	32	1
DirectionQueue	low	low	low	low	1	14	6	1
Game	low	low	low	low-medium	5	17	4	1
Score	low	low	low	low	2	10	6	1
ScoreLabel	low	low-medium	low	low	7	9	7	1
Snake	low-medium	low	low-medium	low-medium	2	30	13	1
Timer	low	low	low	low-medium	1	12	1	1
User	low	low	low	low	0	9	1	1

After:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC
template							
com.snake.game.game	low	low-medium	medium-high	low			
Apple	low	low-medium	low	low	6	28	20
Board	low	low-medium	low	low-medium	6	25	12
Consumable	low	low	low	low	2	2	0
DirectionQueue	low	low	low	low	1	14	6
Game	low-medium	medium-high	medium-high	high	20	98	38
Score	low	low	low	low	2	11	7
ScoreLabel	low	low-medium	low	low	7	10	8
Settings	low	low	low	low	0	16	8
Snake	low-medium	low	low-medium	medium-high	3	35	15

Increasing cohesion for Board

As a result of moving out a big chunk of complexity from Board and moving it to the Game class, the cohesion within the Board class was heavily increased. Since the class did not have to deal with all the different variables that otherwise should have been contained in the game class this made it so that it's cohesion was at an unacceptably low level. As stated before, since we moved many of the functions from Board to Game, this obviously helped increase the overall cohesion and decrease coupling within Board.

Before:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT
template								
com.snake.game.game	low	low-medium	medium-high	low				
Apple	low	low	low	low	4	21	12	1
Board	medium-high	medium-high	medium-high	high	11	90	32	1
DirectionQueue	low	low	low	low	1	14	6	1
Game	low	low	low	low-medium	5	17	4	1
Score	low	low	low	low	2	10	6	1
ScoreLabel	low	low-medium	low	low	7	9	7	1
Snake	low-medium	low	low-medium	low-medium	2	30	13	1
Timer	low	low	low	low-medium	1	12	1	1
User	low	low	low	low	0	9	1	1

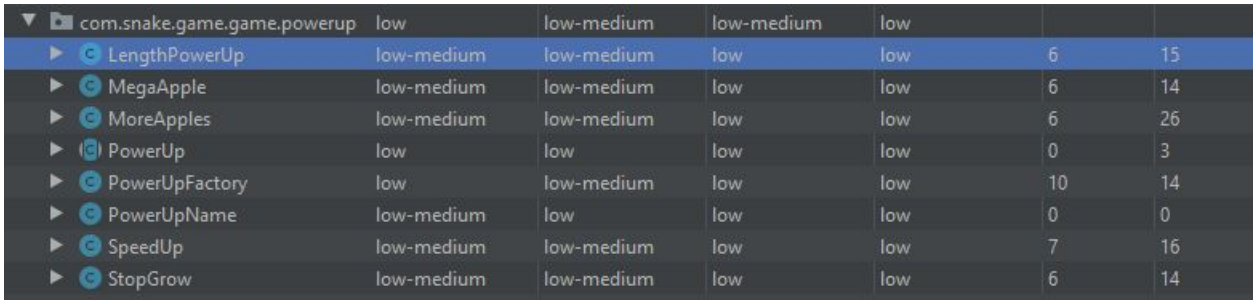
After:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC
template							
com.snake.game.game	low	low-medium	medium-high	low			
Apple	low	low-medium	low	low	6	28	20
Board	low	low-medium	low	low-medium	6	25	12
Consumable	low	low	low	low	2	2	0
DirectionQueue	low	low	low	low	1	14	6
Game	low-medium	medium-high	medium-high	high	20	98	38
Score	low	low	low	low	2	11	7
ScoreLabel	low	low-medium	low	low	7	10	8
Settings	low	low	low	low	0	16	8
Snake	low-medium	low	low-medium	medium-high	3	35	15

PowerUps Coupling refactoring (5 Classes total)

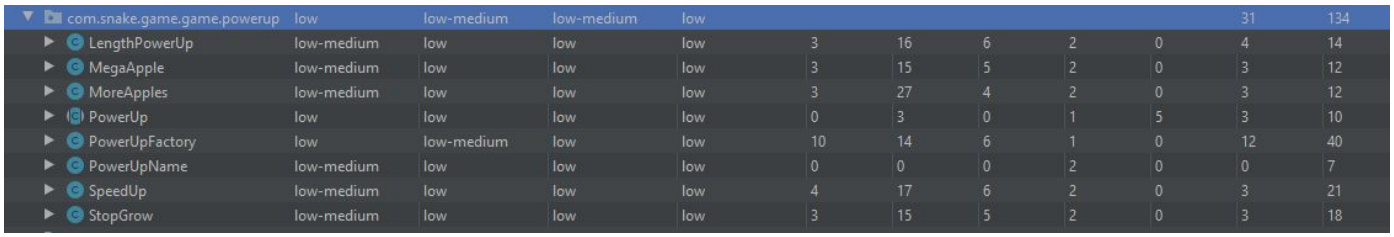
One issue that we faced when refactoring was dealing with couplers (objects/methods causing a high degree of coupling within classes). One particular type of class which faced this issue was the PowerUp class. Each of the powerups unnecessarily had to call the game.getBoard() and draw the powerup on the board using the draw functions. This is what we determined to be the root cause of the elevated coupling in these classes.

Before:



com.snake.game.game.powerup	low	low-medium	low-medium	low		
LengthPowerUp	low-medium	low-medium	low	low	6	15
MegaApple	low-medium	low-medium	low	low	6	14
MoreApples	low-medium	low-medium	low	low	6	26
PowerUp	low	low	low	low	0	3
PowerUpFactory	low	low-medium	low	low	10	14
PowerUpName	low-medium	low	low	low	0	0
SpeedUp	low-medium	low-medium	low	low	7	16
StopGrow	low-medium	low-medium	low	low	6	14

After:



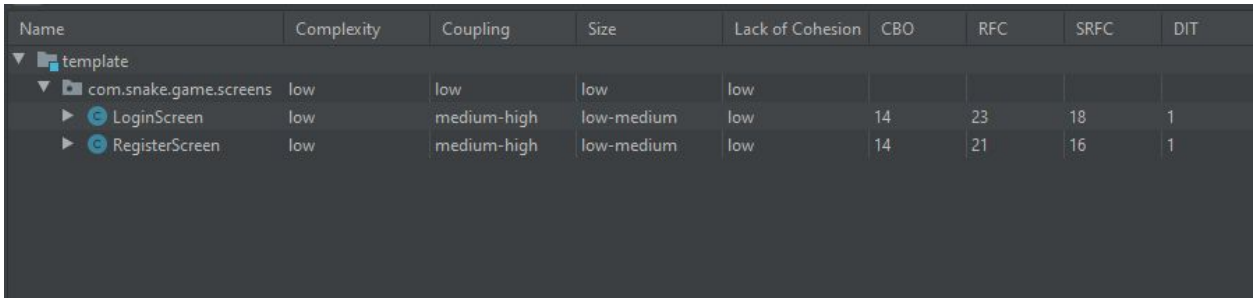
com.snake.game.game.powerup	low	low-medium	low-medium	low						31	134
LengthPowerUp	low-medium	low	low	low	3	16	6	2	0	4	14
MegaApple	low-medium	low	low	low	3	15	5	2	0	3	12
MoreApples	low-medium	low	low	low	3	27	4	2	0	3	12
PowerUp	low	low	low	low	0	3	0	1	5	3	10
PowerUpFactory	low	low-medium	low	low	10	14	6	1	0	12	40
PowerUpName	low-medium	low	low	low	0	0	0	2	0	0	7
SpeedUp	low-medium	low	low	low	4	17	6	2	0	3	21
StopGrow	low-medium	low	low	low	3	15	5	2	0	3	18

To remedy these issues we simply moved the method that is responsible for drawing these Powers all the way to the **Game** class which resulted in a decreased level of coupling since each PowerUp did now not have to manually call the board and draw itself using the board object. In total we changed 5 Power Up Classes

LoginScreen/RegisterScreen

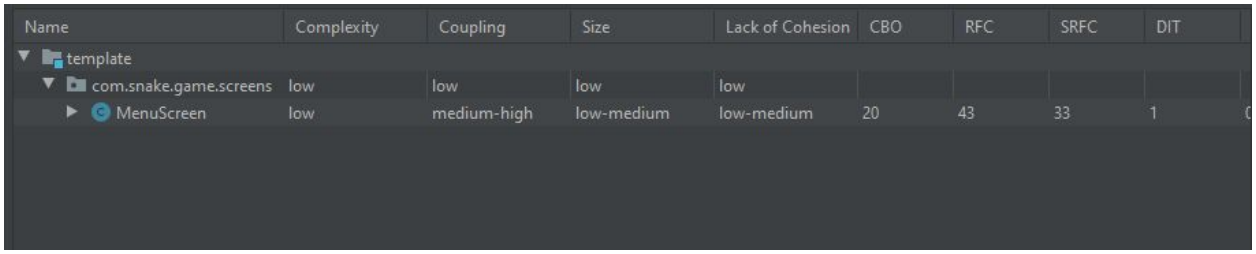
After realizing that the RegisterScreen was basically a copy of LoginScreen, except for the buttons, we attempted to merge those classes into one overarching MenuScreen, which contains two sub-states: Login and Register. We marked the RegisterScreen as '**Dispensable**' (Renaming the LoginScreen to MenuScreen, as stated before). This increased the inter-classical cohesion, but decreased the cohesion within the MenuScreen, due to its extra complexity from adding the states.

Before:



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT
template								
com.snake.game.screens	low	low	low	low				
LoginScreen	low	medium-high	low-medium	low	14	23	18	1
RegisterScreen	low	medium-high	low-medium	low	14	21	16	1

After:



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT
template								
com.snake.game.screens	low	low	low	low				
MenuScreen	low	medium-high	low-medium	low-medium	20	43	33	1

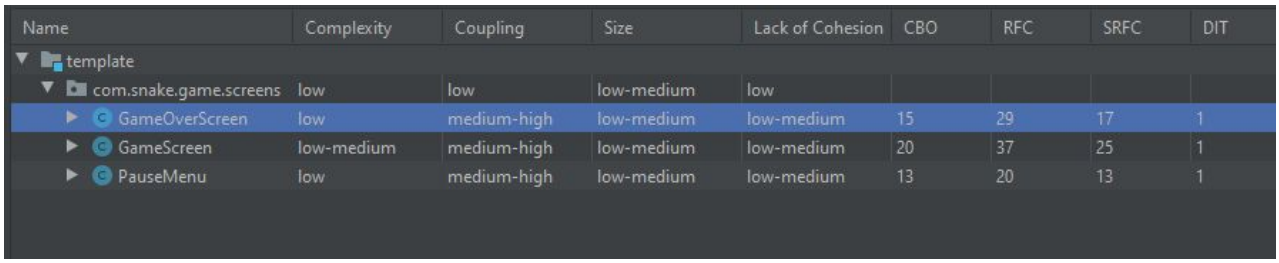
Comments:

The positive effects aren't in the Coupling or LoC, but in the fact that it has been reduced to 1 file.

GameScreen/GameOverScreen/PauseMenu

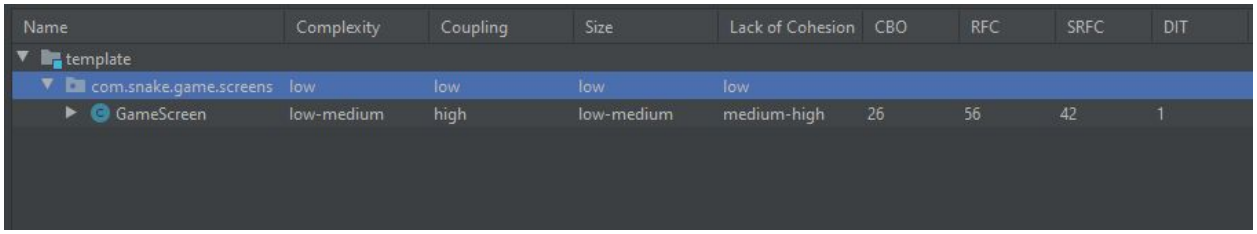
These classes quite unexpectedly contained serious '**Change preventers**' in their screen switching methods. This was due to a long chain of illogical setups: To reset the game (from GameOverScreen) one must be able to open the GameScreen and reset the game field. To resume a game (from PauseMenu) one needed to open the game screen without resetting the game. This means that we somehow needed to pass a parameter (reset or not) to the screen switcher, which would complicate things allot, so we did a work-around, where we would: To resume the game, we just open the GameScreen. To Reset the game, we would construct a new ScreenSwitcher, which would construct a new GameScreen, which would construct a new Game, which would reset the board, and then switch to that GameScreen with the newly created ScreenSwitcher. To prevent this complete mayhem of series of illogical steps, we integrated GameOverScreen and PauseMenu into GameScreen as we did in LoginScreen/RegisterScreen into MenuScreen. Like this, we avoided the need to add an extra parameter to the ScreenSwitcher (which would mostly be unused), or other work-around ways of fixing that issue. The effect of removing the GameScreen and PauseMenu makes it a 'Dispensable'

Before:



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT
template								
com.snake.game.screens	low	low	low-medium	low				
GameOverScreen	low	medium-high	low-medium	low-medium	15	29	17	1
GameScreen	low-medium	medium-high	low-medium	low-medium	20	37	25	1
PauseMenu	low	medium-high	low-medium	low-medium	13	20	13	1

After:



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT
template								
com.snake.game.screens	low	low	low	low				
GameScreen	low-medium	high	low-medium	medium-high	26	56	42	1

Comments:

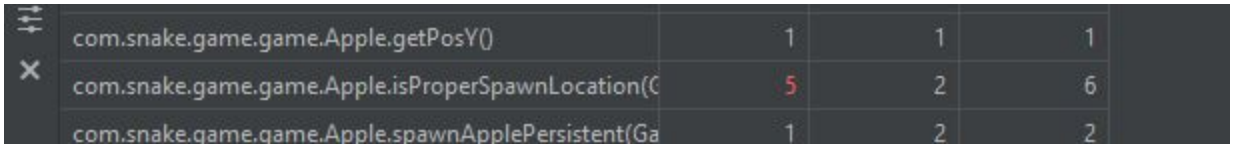
The positive effects aren't in the Coupling or LoC, but in the fact that it has been reduced to 1 file.

5 Method-Level Code Smell Refactors

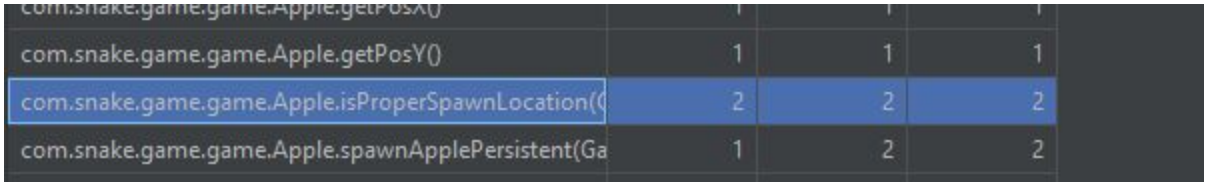
Apple.isProperSpawnLocation Complexity Reduction

The isProperSpawnLocation method within Apple after our analysis did in fact turn out to be unnecessarily complicated and convoluted. The essential cyclomatic complexity of the method was simply too high and made the method difficult to maintain and modify.

Before:



After:

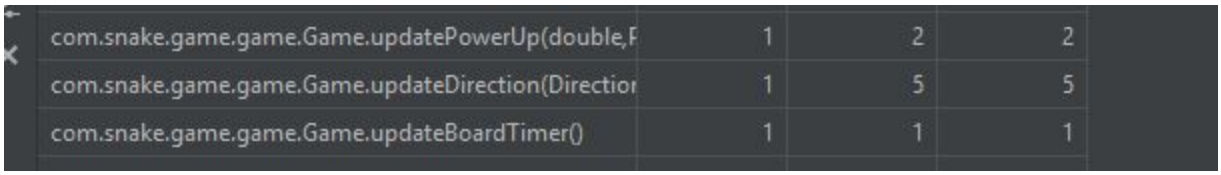


To get around this issue we decided to refactor the method into several different methods each which had a different responsibility. Instead of just having one method to check for all the possible collisions of an apple, this work was now subdivided into several different classes, helping to decrease the essential cyclomatic complexity of this method increasing it’s maintainability and stability.

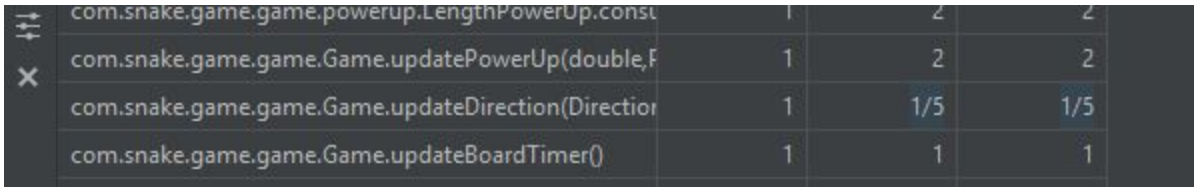
Game.updateDirection(Snake.Direction direction)

During our refactoring process we found a method which might now have had a very high cyclomatic complexity, but the cyclomatic complexity that was within the method was completely unreasonable. Instead of having a length few lines of if statements we simply refactored the method to have 1 line of code that updates the direction.

Before:



After:



Signup.execute() Complexity Reduction

Before:

method	ev(G)	iv(G)	v(G)
com.snake.game.requests.Signup.execute()	5	7	7
com.snake.game.requests.Login.execute()	4	6	6
com.snake.game.requests.MaxScore.execute()	3	5	5
com.snake.game.requests.UserInfo.execute()	3	5	5
com.snake.game.game.Snake.collides(int,int)	3	3	4
com.snake.game.game.Apple.checkApplesCollision(Game,int,	3	1	4
com.snake.game.game.Timer.timerHandler(long)	2	2	5
com.snake.game.game.Apple.isProperSpawnLocation(Game,ir	2	2	2

After:

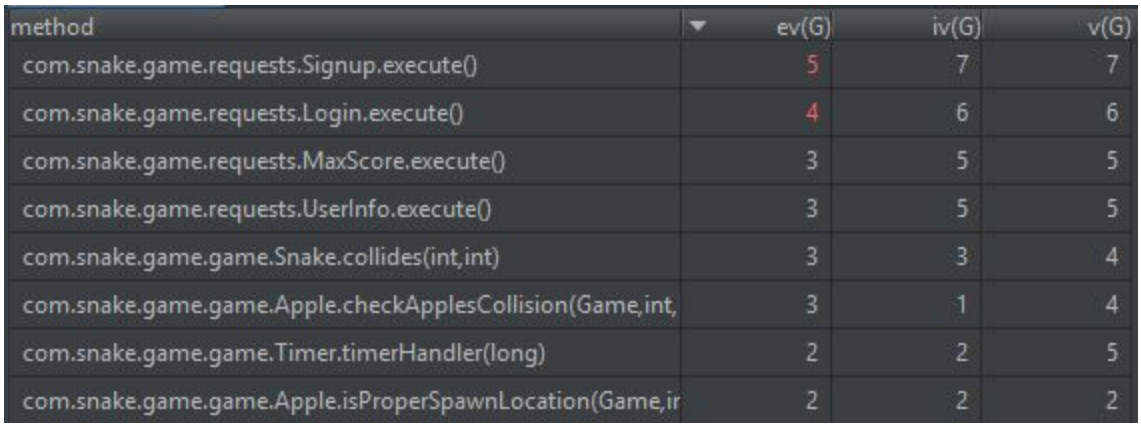
com.snake.game.game.Apple.checkApplesCollision(Game,int,int)	3	1	4
com.snake.game.requests.Signup.execute()	3	3	4
com.snake.game.requests.Signup.usernameCheck()	2	3	3

Comments:

As high essential cyclomatic complexity can be fixed through refactoring code using polymorphism, in this example it would not make much sense, so the best choice was to create two different method that check correctness of password and username.

Login.execute() Complexity Reduction

Before:



method	ev(G)	iv(G)	v(G)
com.snake.game.requests.Signup.execute()	5	7	7
com.snake.game.requests.Login.execute()	4	6	6
com.snake.game.requests.MaxScore.execute()	3	5	5
com.snake.game.requests.UserInfo.execute()	3	5	5
com.snake.game.game.Snake.collides(int,int)	3	3	4
com.snake.game.game.Apple.checkApplesCollision(Game,int,	3	1	4
com.snake.game.game.Timer.timerHandler(long)	2	2	5
com.snake.game.game.Apple.isProperSpawnLocation(Game,ir	2	2	2

After:

com.snake.game.requests.Login.execute()	3	3	4

Comments:

This refactoring has similar impact and reasons as Signup.execute() method.

Game.run()

method	ev(G)	iv(G)	v(G)
com.snake.game.game.states.ActiveGameState.keyPress()	1	11	11
com.snake.game.game.Game.run()	2	8	8
com.snake.game.screens.MenuScreen.addListeners()	1	8	8

Comments:

This method was fairly long and therefore unreadable. We divided it into 4 separate methods with meaningful names. This resulted in an easier to maintain method but the game class did become a bit more overloaded with all these additional methods. This however should be fine since game class does act as the major component tying in all the components of the game together.

method	ev(G)	iv(G)	v(G)
com.snake.game.game.DirectionQueue.dequeue()	1	2	2
com.snake.game.game.Apple.isProperSpawnLocation(Game,ir	2	2	2
com.snake.game.game.Game.run()	2	1	2