# Assignment 4 : Exercise 2

**Part 1 :**
For code metrics of methods we used Metrics Reloaded
For code metrics of classes we used Code MR

ECC = essential cyclomatic complexity
CC = cyclomatic complexity

**Methods that need refactoring :**

- runGame(int) in Game class → ECC = 5 , CC = 6
- createUser(String, String) in UserController class → ECC = 4 , CC = 7
- onMessage(String) in MatchSocketHandler class → ECC = 2, CC = 9
- paintComponent(Graphics) in Field class → ECC = 1, CC = 8
- goalCollision(Frame) in Puck class → ECC = 1, CC = 7

We need to refactor these methods because the Metric Reloaded tool showed these methods as the first 2 exceeded the metric thresholds. The essential cyclomatic complexity on these methods was too high and these methods. The essential cyclomatic complexity tells us that there are many breaks and return statements in structured components such as loops. These methods are therefore harder to refactor into multiple methods. The last 3 methods have a too high cyclomatic complexity and also could use some refactoring. This would mean you have to have 9 tests to have 100% branch coverage on the onMessage method, this is too much.

**runGame:** The runGame method had a higher complexion because of 2 return statements in a for loop at the end of the method to update the board every frame. So this will be split up into its own method to be called.

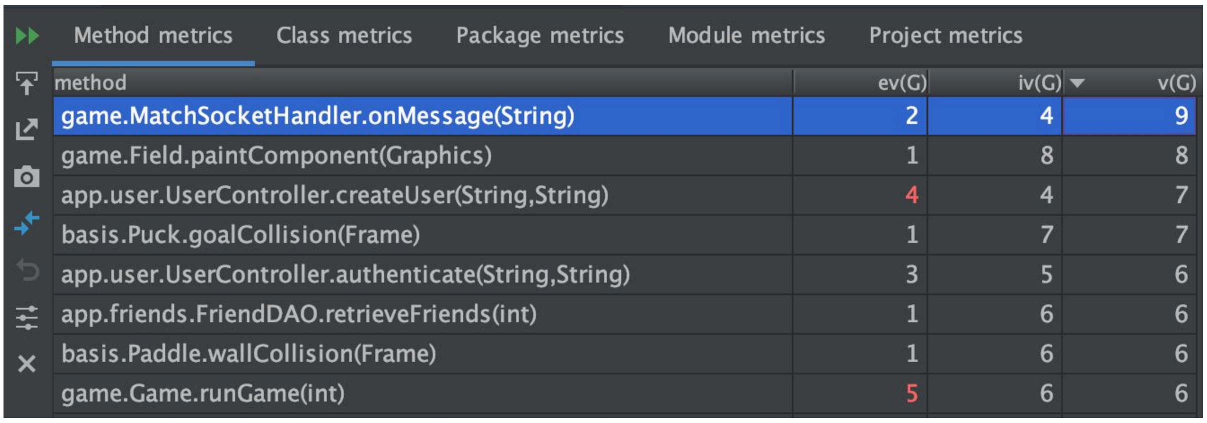
**Classes that need refactoring :**

- Puck class, complexity is medium-high
- MovingEntity class, complexity is medium-high
- Paddle class, complexity is medium-high
- MatchSocketHandler class, coupling and lack of cohesion is medium-high
- Frame class, complexity is medium-high

These classes are of the few classes that are marked medium-high complexity and coupling in our src folder by the Code MR plugin, therefore we should refactor these methods. Mainly the complexity is a recurring code smell we should refactor.

**<u>Part 2 :</u>**

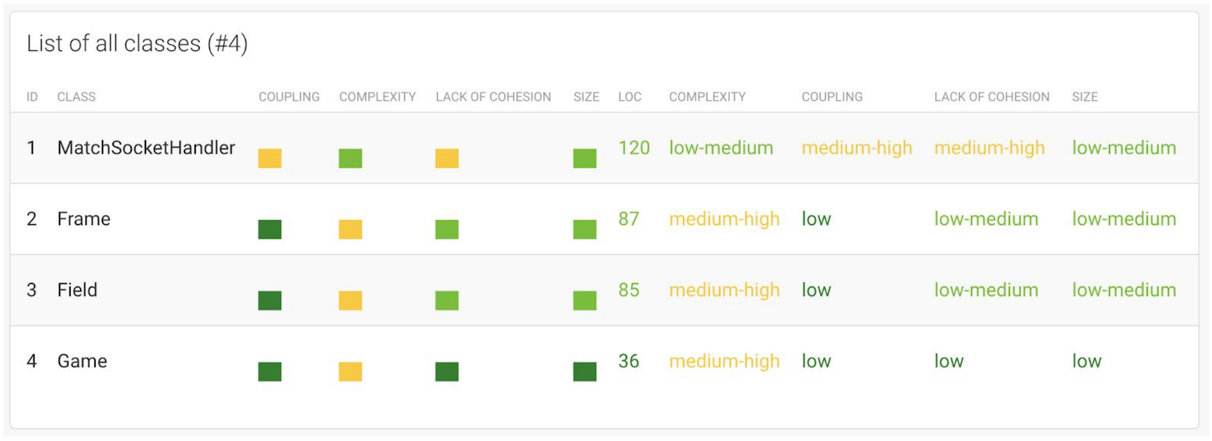
These are the code metrics we generated **before** refactoring the 5 classes and methods.



Method metrics | Class metrics | Package metrics | Module metrics | Project metrics

| method | ev(G) | iv(G) | v(G) |
|---|---|---|---|
| game.MatchSocketHandler.onMessage(String) | 2 | 4 | 9 |
| game.Field.paintComponent(Graphics) | 1 | 8 | 8 |
| app.user.UserController.createUser(String,String) | 4 | 4 | 7 |
| basis.Puck.goalCollision(Frame) | 1 | 7 | 7 |
| app.user.UserController.authenticate(String,String) | 3 | 5 | 6 |
| app.friends.FriendDAO.retrieveFriends(int) | 1 | 6 | 6 |
| basis.Paddle.wallCollision(Frame) | 1 | 6 | 6 |
| game.Game.runGame(int) | 5 | 6 | 6 |

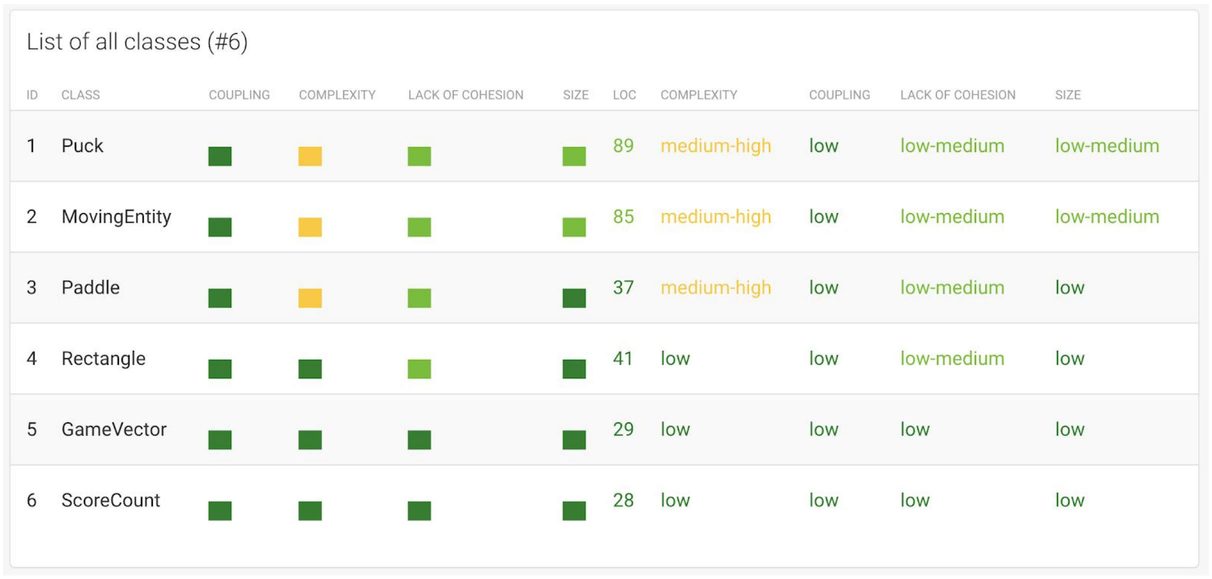
*Metrics Reloaded plugin



List of all classes (#4)

| ID | CLASS | COUPLING | COMPLEXITY | LACK OF COHESION | SIZE | LOC | COMPLEXITY | COUPLING | LACK OF COHESION | SIZE |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | MatchSocketHandler | 🟨 | 🟩 | 🟨 | 🟩 | 120 | low-medium | medium-high | medium-high | low-medium |
| 2 | Frame | 🟩 | 🟨 | 🟩 | 🟩 | 87 | medium-high | low | low-medium | low-medium |
| 3 | Field | 🟩 | 🟨 | 🟩 | 🟩 | 85 | medium-high | low | low-medium | low-medium |
| 4 | Game | 🟩 | 🟨 | 🟩 | 🟩 | 36 | medium-high | low | low | low |

*Code MR plugin



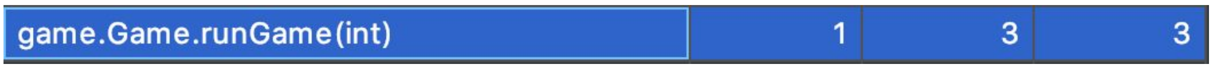
List of all classes (#6)

| ID | CLASS | COUPLING | COMPLEXITY | LACK OF COHESION | SIZE | LOC | COMPLEXITY | COUPLING | LACK OF COHESION | SIZE |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Puck | 🟩 | 🟨 | 🟩 | 🟩 | 89 | medium-high | low | low-medium | low-medium |
| 2 | MovingEntity | 🟩 | 🟨 | 🟩 | 🟩 | 85 | medium-high | low | low-medium | low-medium |
| 3 | Paddle | 🟩 | 🟨 | 🟩 | 🟩 | 37 | medium-high | low | low-medium | low |
| 4 | Rectangle | 🟩 | 🟩 | 🟩 | 🟩 | 41 | low | low | low-medium | low |
| 5 | GameVector | 🟩 | 🟩 | 🟩 | 🟩 | 29 | low | low | low | low |
| 6 | ScoreCount | 🟩 | 🟩 | 🟩 | 🟩 | 28 | low | low | low | low |

*Code MR plugin

These are the code metrics we generated **after** refactoring the 5 classes and methods:

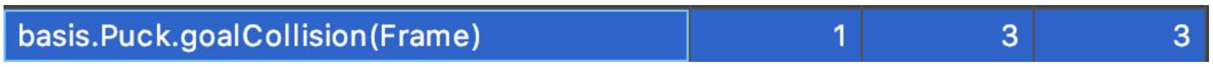
runGame(int) in Game class:



| | | | |
|---|---|---|---|
| game.Game.runGame(int) | 1 | 3 | 3 |

To decrease the complexity of runGame I have split it up in a part that updates the screen every frame in runGame and a part that can be called every time to move the puck in movePuck.

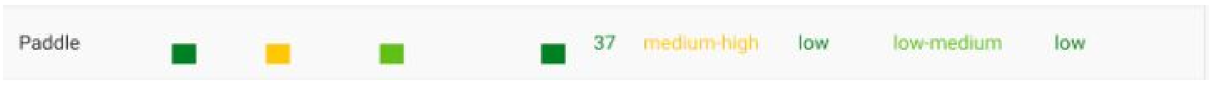
goalCollision(Frame) in Puck class:



| | | | |
|---|---|---|---|
| basis.Puck.goalCollision(Frame) | 1 | 3 | 3 |

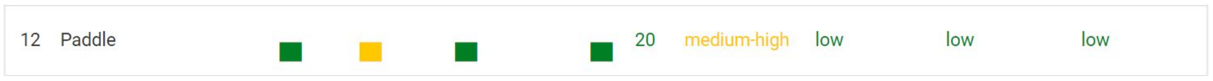
To decrease the cyclomatic complexity of this method, we separated the long conditions and made boolean methods out of them.

Paddle:
Before:



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Paddle | 🟩 | 🟨 | 🟩 | 🟩 | 37 | medium-high | low | low-medium | low |

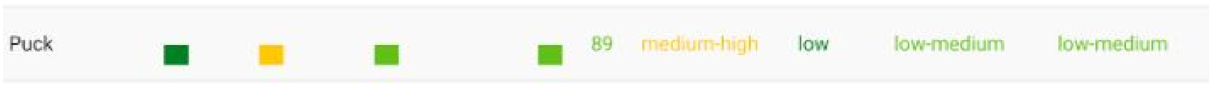
After:



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 12 | Paddle | 🟩 | 🟨 | 🟩 | 🟩 | 20 | medium-high | low | low | low |

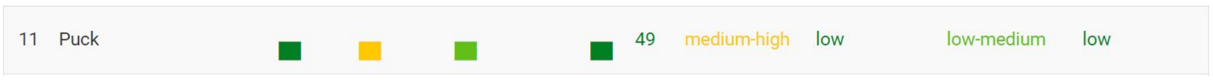
In order to decrease the lack of cohesion, we moved the wall collisions to an external class called Bounds.

Puck:
Before:



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Puck | 🟩 | 🟨 | 🟩 | 🟩 | 89 | medium-high | low | low-medium | low-medium |

After:



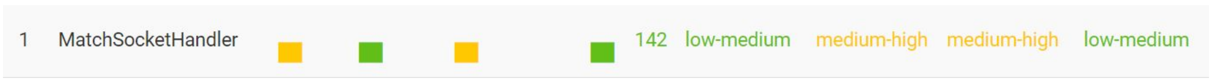
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 11 | Puck | 🟩 | 🟨 | 🟩 | 🟩 | 49 | medium-high | low | low-medium | low |

In order to decrease the size of the Puck class, we moved the goal collisions to an external class called Bounds.

MatchSocketHandler:
Moved methods that generate paddle, puck and score values to be sent to server to those respective classes or to the Frame method, which deals with match specific instances of

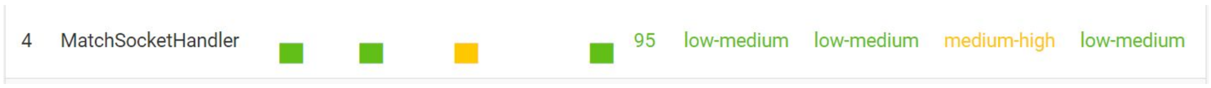
those classes. This should limit coupling of class to the Frame class and the ScoreCount class and it shouldn't need to interact with pucks or paddles.

Before



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | MatchSocketHandler | 🟨 | 🟩 | 🟨 | 🟩 | 142 | low-medium | medium-high | medium-high | low-medium |

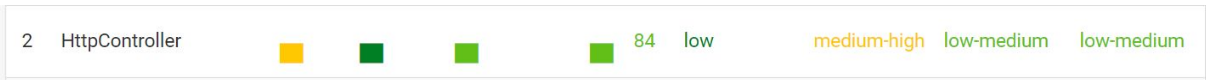
After



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | MatchSocketHandler | 🟩 | 🟩 | 🟨 | 🟩 | 95 | low-medium | low-medium | medium-high | low-medium |

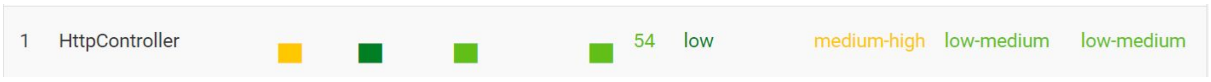
HttpController:
Simplified the generation and processing of http requests and reduced lines of code in class. Couldn't reduce coupling further because it the single point of HTTP request generation for all other controllers.

Before



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | HttpController | 🟨 | 🟩 | 🟩 | 🟩 | 84 | low | medium-high | low-medium | low-medium |

After



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | HttpController | 🟨 | 🟩 | 🟩 | 🟩 | 54 | low | medium-high | low-medium | low-medium |

We weren't able to refactor all the classes and methods we pointed out. This is because we had very few classes that needed refactoring. We thought we could improve on these methods and classes but when trying, we figured out that it would only break the code instead of making it better. This is similar to the 100% code coverage fallacy, you shouldn't only strive for a green color on every class in the Code MR reports. If your classes are well refactored and refactoring them even further can go at a cost of breaking or slowing down your game then it is a sign that your code is good as it is. I think a reason for this probably is that we refactored our code during the whole project.