Team 42 Air Hockey
Robert Minea
Ionut Constantinescu
Jaron Rosenberg
Darwin Morris

Refactoring and wrap-up

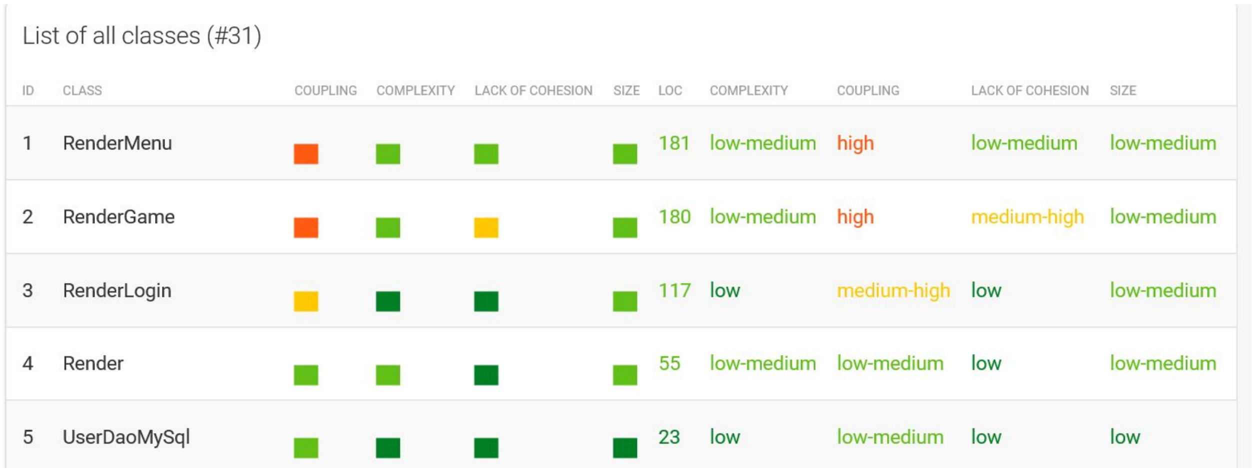
Completing the game

For the last assignment we have added the finishing touches in order to make sure that our game is playable and that all the must haves are implemented. The following functionalities/fixes/improvements have been added:

- The user can now change his nickname in the menu screen
- The game history can be found in the details section of the menu
- Now the puck is being generated randomly on the board at the beginning of the game (either on the left or right side of the center)
- The pusher positions are reset after a goal
- The number of the player is now being displayed in the corner of the screen (left and right), the first logged in user is player 1, and the second is player 2
- Solved a bug related to updating the user statistics
- Now only one connection is being created to the database when the application is started
- Leaderboard position in menu details

Identifying The Problems using an analysis tool

The CodeMR Code Quality Analysis of the 5 most coupled classes and their methods from the master at the start of this sprint.



List of all classes (#31)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	RenderMenu	🟥	🟩	🟩	🟩	181	low-medium	high	low-medium	low-medium
2	RenderGame	🟥	🟩	🟨	🟩	180	low-medium	high	medium-high	low-medium
3	RenderLogin	🟨	🟩	🟩	🟩	117	low	medium-high	low	low-medium
4	Render	🟩	🟩	🟩	🟩	55	low-medium	low-medium	low	low-medium
5	UserDaoMySql	🟩	🟩	🟩	🟩	23	low	low-medium	low	low

It seems like the main problems are caused by the RenderStrategies (and also the User class [not seen in picture])

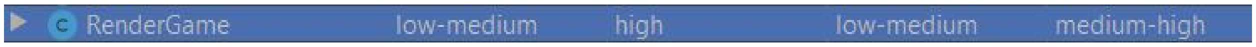
Refactored classes

RenderGame class

This class was one of the most problematic classes, suffering from blob code and and high coupling

Before:

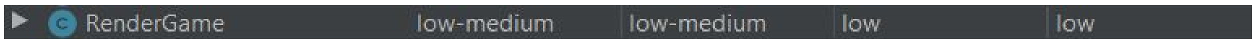
CBO 28, LOC 180, LCOM 0.88 (from left to right Complexity, Coupling, Size, Lack of Cohesion)



RenderGame	low-medium	high	low-medium	medium-high

After:

CBO 9, LOC 31, LCOM 0.5



RenderGame	low-medium	low-medium	low	low

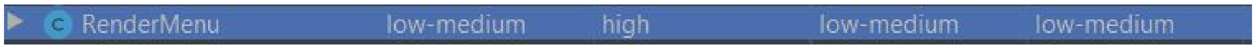
- In order to solve this, we have mitigated the functionalities of this class by placing the draw functions and database interactions in other classes.

- The scoreboard is now the one who interacts with the database (the drawTopScores and uploadGame functions have been moved here) since this object holds the score and it knows when the game is going to end.
- Now the drawing of the objects, background and text is not being done in the renderGame anymore, the Puck and Pusher will have to make sure that they will draw themselves after every game iteration, and now the information and board of the match are being drawn by calling the functions of ObjectDrawer and InformationDrawer.
- Also the match doesn't handle the update of every element now since a container of the match has been created as the Match class, and now the RenderGame only has to call the methods of this class in order to advance the game match.

RenderMenu class

Before:

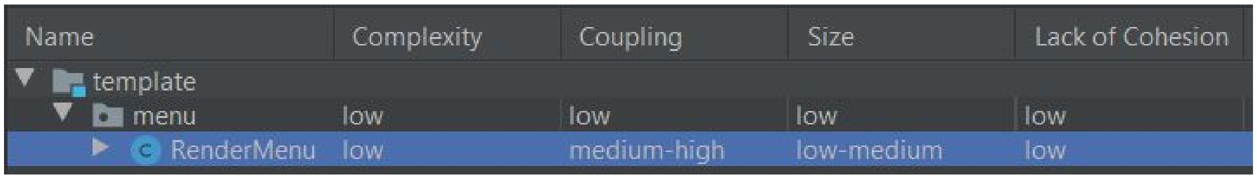
CBO 27, 0.842 LCOM, 181 LOC (from left to right Complexity, Coupling, Size, Lack of Cohesion)



RenderMenu	low-medium	high	low-medium	low-medium

After:

CBO 20, 0.815 LCOM, 80 LOC



Name	Complexity	Coupling	Size	Lack of Cohesion
template				
menu	low	low	low	low
RenderMenu	low	medium-high	low-medium	low

- After the last features of the game has been implemented, the RenderMenu class has been deemed as a problematic class by the CodeMR tool, as it was suffering from high coupling due to the large number of classes that it was relying on, we also managed to lower its complexity after the refactoring.

- We have mitigated this high coupling by creating a container for the MenuButtons that is in charge of rendering the images corresponding to these buttons.

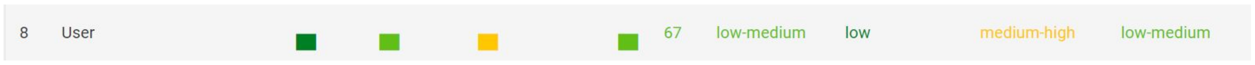
GameObject class

- The GameObject class has been created after pondering on the fact that both the Pusher and the Puck share most of their fields since they are very similar in mechanics. Having them stored in a centralized place seemed like a good idea, so the GameObject class has been created as a super class for the Puck and Pusher. The LOC is now lower with approx 10 lines and the structure of these classes is now a lot clearer.

User class

Before:

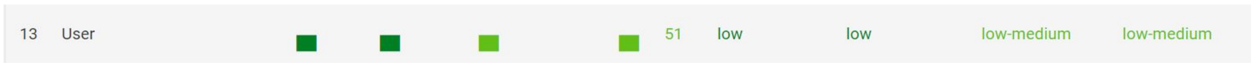
CBO: 1, LCOM: 0.731 , LOC: 67, WMC: 22 , RFC: 22



8	User	🟩	🟩	🟨	🟩	67	low-medium	low	medium-high	low-medium

After:

CBO: 0, LCOM: 0.667 , LOC: 51, WMC: 16, RFC: 15



13	User	🟩	🟩	🟩	🟩	51	low	low	low-medium	low-medium

- This class was suffering of lack of cohesion between methods.
- We removed unused methods, remove game history list.
- We refactored some methods to include multiple functionalities (merge addLostGame and SetLostGames into a single method).
- LCOM is mostly due to the fact that we have the fields for the user object as represented in the database and getters and setters create low cohesion score (no reason to improve further, it can make the code harder to understand).

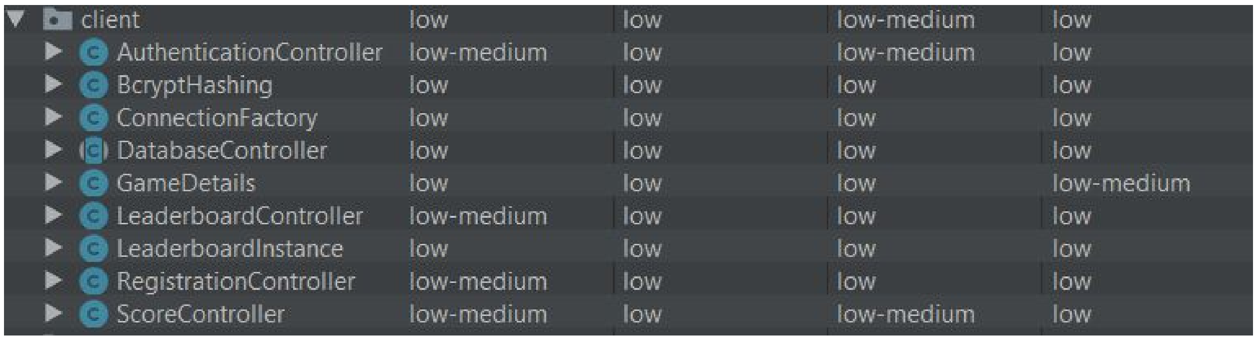
Dao design pattern (UserDao, LeaderboardDao, UserGameTracker, UserAuthentication, UserRegistration)

The Data Access Object (DAO) pattern is a structural pattern that allows us to isolate the application/business layer from the persistence layer (usually a relational database, but it could be any other persistence mechanism) using an abstract API.

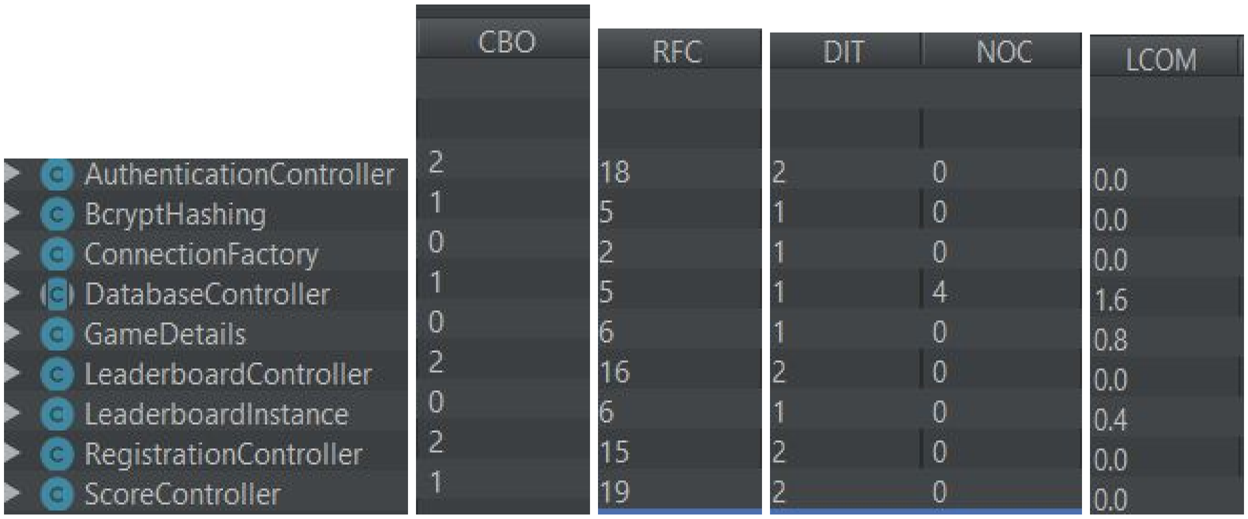
The functionality of this API is to hide from the application all the complexities involved in performing CRUD operations in the underlying storage mechanism. This permits both layers to evolve separately without knowing anything about each other.

There are just slight/minor improvements regarding the metrics but the bigger improvement is in the object design and code modularity.

Before

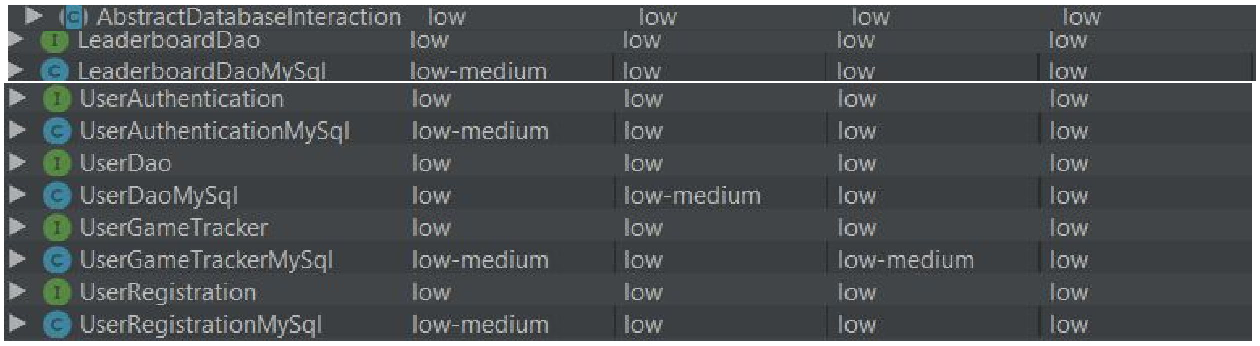


client	low	low	low-medium	low
AuthenticationController	low-medium	low	low-medium	low
BcryptHashing	low	low	low	low
ConnectionFactory	low	low	low	low
DatabaseController	low	low	low	low
GameDetails	low	low	low	low-medium
LeaderboardController	low-medium	low	low	low
LeaderboardInstance	low	low	low	low
RegistrationController	low-medium	low	low	low
ScoreController	low-medium	low	low-medium	low

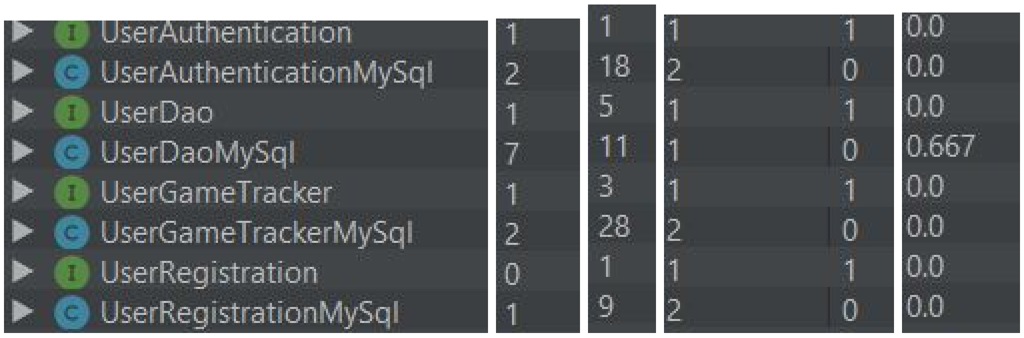


	CBO	RFC	DIT	NOC	LCOM
AuthenticationController	2	18	2	0	0.0
BcryptHashing	1	5	1	0	0.0
ConnectionFactory	0	2	1	0	0.0
DatabaseController	1	5	1	4	1.6
GameDetails	0	6	1	0	0.8
LeaderboardController	2	16	2	0	0.0
LeaderboardInstance	0	6	1	0	0.4
RegistrationController	2	15	2	0	0.0
ScoreController	1	19	2	0	0.0

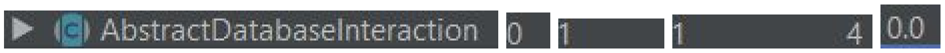
After



AbstractDatabaseInteraction	low	low	low	low
LeaderboardDao	low	low	low	low
LeaderboardDaoMySql	low-medium	low	low	low
UserAuthentication	low	low	low	low
UserAuthenticationMySql	low-medium	low	low	low
UserDao	low	low	low	low
UserDaoMySql	low	low-medium	low	low
UserGameTracker	low	low	low	low
UserGameTrackerMySql	low-medium	low	low-medium	low
UserRegistration	low	low	low	low
UserRegistrationMySql	low-medium	low	low	low



UserAuthentication	1	1	1	1	0.0
UserAuthenticationMySql	2	18	2	0	0.0
UserDao	1	5	1	1	0.0
UserDaoMySql	7	11	1	0	0.667
UserGameTracker	1	3	1	1	0.0
UserGameTrackerMySql	2	28	2	0	0.0
UserRegistration	0	1	1	1	0.0
UserRegistrationMySql	1	9	2	0	0.0



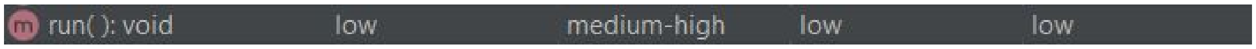
AbstractDatabaseInteraction	0	1	1	4	0.0

Refactored Methods

RenderGame - run() method

Before

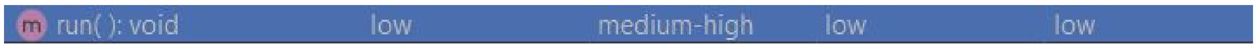
CBO 8 LOC 26 (from left to right Complexity, Coupling, Size, Lack of Cohesion)



run( ): void	low	medium-high	low	low

After

CBO 9 LOC 12



run( ): void	low	medium-high	low	low

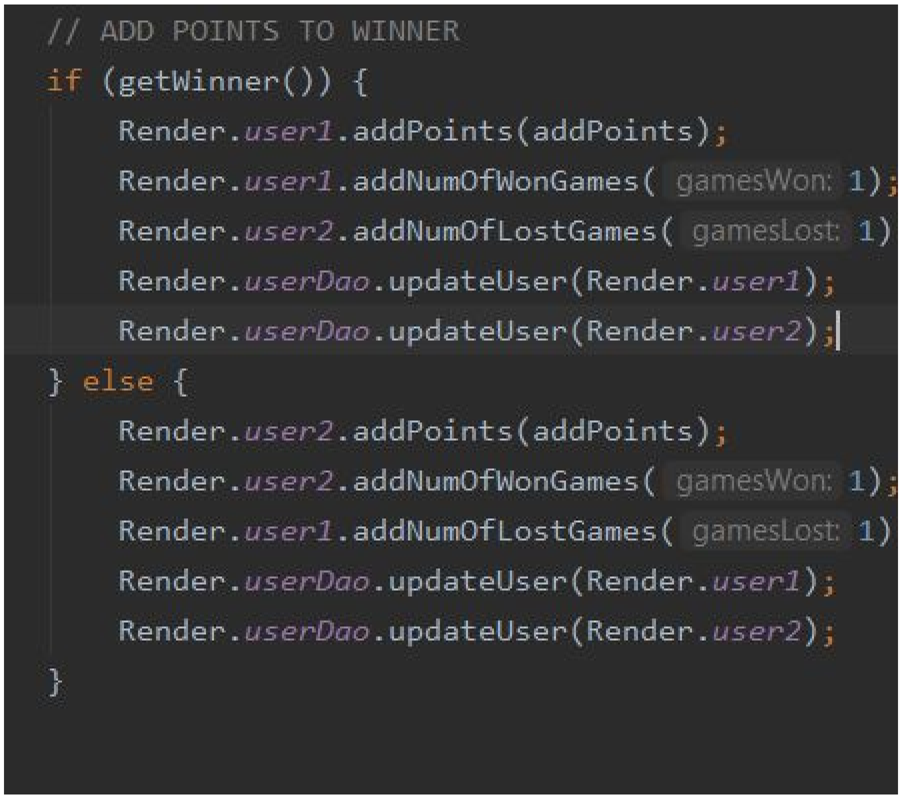
- The run method of the game was one of the most problematic functions in our entire application since it was suffering from high coupling and blob code. Many functions and functionalities were being clustered together in this application and this is why we have decided to better hand the functionalities of this method to other classes where the calls seem to be a better fit.
- As mentioned earlier we have created the ObjectDrawer and InformationDrawer classes in order to render all the static graphical elements on the screen, and now only a few calls are necessary in order to render the whole screen.

- One other improvement is that we have assigned the objects with the task of drawing their own graphical representation after updating their own positions, This seemed like a good way to mitigate the high coupling.
- The last thing that we have done is create the Match class that now acts as a container for the game Match itself and calls all the necessary functions in order to update the state of the game.

ScoreBoard - uploadMatch() method

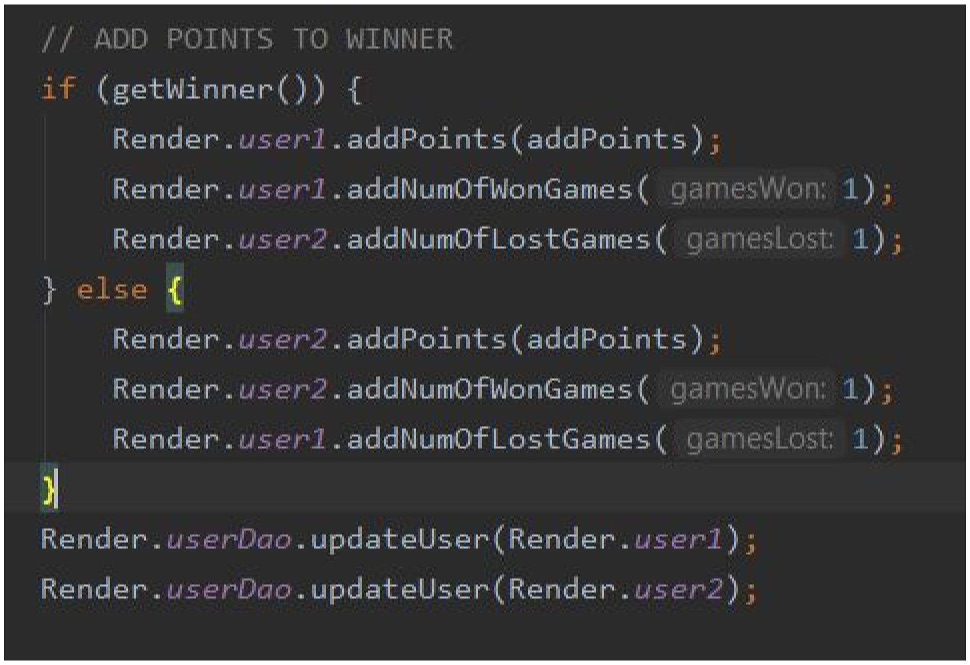
- After checking the code we have noticed that in the upload match class there are 4 different calls to the updateUser method. Although not a big problem, we can cut 2 calls since they are of no use, now also making the code more readable, reducing the LOC from 16 to 14.

Before



```
// ADD POINTS TO WINNER
if (getWinner()) {
    Render.user1.addPoints(addPoints);
    Render.user1.addNumOfWonGames( gamesWon: 1);
    Render.user2.addNumOfLostGames( gamesLost: 1)
    Render.userDao.updateUser(Render.user1);
    Render.userDao.updateUser(Render.user2);
} else {
    Render.user2.addPoints(addPoints);
    Render.user2.addNumOfWonGames( gamesWon: 1);
    Render.user1.addNumOfLostGames( gamesLost: 1)
    Render.userDao.updateUser(Render.user1);
    Render.userDao.updateUser(Render.user2);
}
```

After

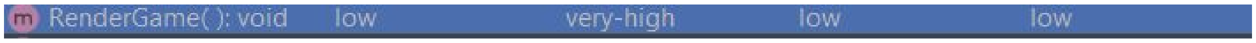


```
// ADD POINTS TO WINNER
if (getWinner()) {
    Render.user1.addPoints(addPoints);
    Render.user1.addNumOfWonGames( gamesWon: 1);
    Render.user2.addNumOfLostGames( gamesLost: 1);
} else {
    Render.user2.addPoints(addPoints);
    Render.user2.addNumOfWonGames( gamesWon: 1);
    Render.user1.addNumOfLostGames( gamesLost: 1);
}
Render.userDao.updateUser(Render.user1);
Render.userDao.updateUser(Render.user2);
```

RenderGame - RenderGame() constructor

Before

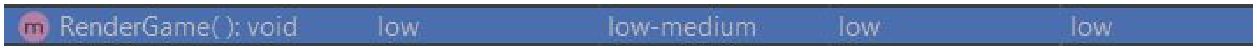
CBO 13, LOC 20 (from left to right Complexity, Coupling, Size, Lack of Cohesion)



RenderGame( ): void	low	very-high	low	low

After

CBO 4, LOC 5



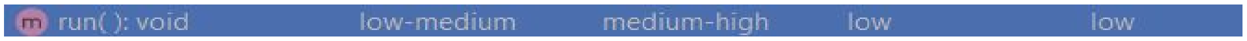
RenderGame( ): void	low	low-medium	low	low

- The RenderGame constructor is another method that was suffering from high coupling, containing all the object instantiations for all the elements necessary in the match. This might have been too much, and this is why we have delegated the task of drawing to the specific drawers and instantiating/updating of the GameObjects and details to the Match class.

RenderMenu - run() method

Before

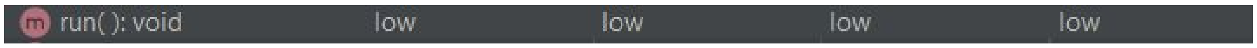
CBO 8, LOC 24



run( ): void	low-medium	medium-high	low	low

After

CBO 2, LOC 6



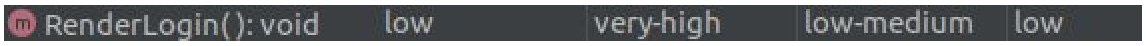
run( ): void	low	low	low	low

- This method had many lines of code and high coupling, being very problematic. In order to make it easier to use and understand, we have grouped all the initialization of the images (of the buttons) from the menu into the MenuButtons class, which manages the interaction with these buttons and just called its method here for interacting with them

RenderLogin - RenderLogin() method

Before

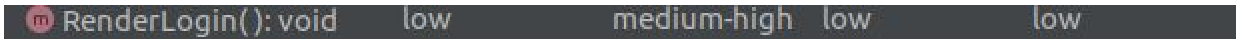
CBO 13, LOC 91



RenderLogin( ): void	low	very-high	low-medium	low

After

CBO 7, LOC 32



RenderLogin( ): void	low	medium-high	low	low

- Previously this method handled all the functionality of initializing the actors that were used in order to render the login screen. This lead to a very coupled and long method. This was solved by creating separate classes to initialize the actors that made up both the text fields as well as the register buttons. These could then simply be accessed as array lists of actors and added to the stage in the RenderLogin() method.