Mehmet Mert Bezirgan - FallingObjectFactory Test

In this test the objective is to test fundamental functionality of FallingObjectFactory using glass box testing. I used glass box testing because there were multiple scenarios that depend on the input. I wrote 5 test cases in which we are testing the getInstance method, creation of power ups, reaction blockers and molecules and finally the coordinates of the objects created. All of the developments were made at the branch testFallingObjectFactory branch and merged to master branch when the development finished.

Doğa Demirtürk - Shooter Test

Main functionalities of the Shooter are to move and to rotate. First of all I wrote a setter-getter test and a constructor test. Then I wrote a move and rotate test using Black Box and Glass Box Testing to cover all cases for move and rotate functionalities. Using Black Box testing, I checked the boundaries of the actions. I tested the move method when the shooter is on the left and right boundaries of the screen. Then I added Glass Box tests for the cases where the shooter is not on the boundaries of the right and left move. Since boundary cases were handled by Black Box Testing, the two tests, one is when the shooter is on the left boundary and moves right and the other is when the shooter is on the right boundary and moves left, are enough to satisfy Glass Box Testing. I tested the rotate functionality the same way I tested the move functionality. Using Black Box testing, I tested the cases when the shooter angle is at the limits of the rotation of the shooter. Then I added two more cases, when the shooter is not on the right limit of the rotate and to rotate right and when the shooter is not on the left limit of the rotate and rotates left, to satisfy the Glass Box Testing. Additionally, I tested the getBarrelCoordinates method which calculates depending on the coordinate of the shooter and its angle and returns the coordinates of the barrel of the shooter. I added necessary specifications to the Shooter class. I worked on branch shooterTest/doga on git and merged into master branch the changes when I finished testing.

İrem Şahin - Eta Shield Test

In this test, I have used Glass Box testing, and I have aimed for the full coverage of the Eta Shield class. I have used Glass Box testing because in the user end of the program, there is only one operation done(add shield from the button) and the main operations of the shield classes are dependent on the internal implementation of the code. I wrote 6 methods for testing, first one being the setUp method that initialized some structures to avoid code repetition in the latter methods, and 5 tests that cover the whole EtaShield class methods(addShield, getEfficiency,setEfficiency,constructor and decorator pattern). The tests show how the methods generally work, along with how the overriding some methods affect the outcome of the getter methods. All code development is done in the test/etashieldtest branch and uploaded to there.

Ceyhun Aslan - Beta Molecule Test

In this test, I used Black Box testing for testing the BetaMolecule class. The tests first test the constructor and the getters setters of the class by primarily using assertEquals. After the basic test, the class tests the falling functionality of the BetaMolecule with three different tests. The BetaMolecule changes its fallingBehaviour by checking the height and has a specific breakpoint. The first test checks the fallingBehaviour before and after the break point with the updateFallingBehaviour method. The last two tests test the moving functionality in the two different fallingBehaviour. These developments were done in the testBetaMolecule branch.

Yarkın GAZİ - Player Test

I have done testing on Player Class. In this testing I have used Black Box testing. Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing. There are 4 functions in the player class which are: increaseScore(double points), decreaseHealth(double health), getScore(), getHealth(). I have applied 5 testing implementations. Each for one testing and 2 for getScore. To this aim, I have imported org.junit.jupiter.api.Test.