Evaluating code of others

Source code quality:

- Attributes in player are set to public. It's better to set it to private and create getters and setters.
- The classes winScene and gameoverSceen are almost the same. This could be improved by making a super class that has the duplication methods. Or create a gameDoneclass with a string input for the png. Because the png string is the only difference in the classes.
- The methods doupdate in the ball class and drop class have the same code. Try to make a superclass to optimize the code.
- Lifesystem, soundSystem and scoresystem should be set to final.
- Coburta does not run with the pom.xml.This should be implemented because then the coverage of the test are available.

Design patterns are well placed. The patterns that are used makes sense for those problems. Other software engineering principles are oke.

Grade: 7

Testing:

- The dropRandomizer test is not finished. Now it's only there to improve coverage but not to ensure code quality because there are now asserts. There is some code that could be tested.
- There many method tests that have multiple asserts. This is not wrong but not easy to handle when they fail. Because the tests do not show which assert has failed. Try to split these test in to smaller test and only assert once each test method.

There are a lot of tests with different parameters to test most branches. Good use of the setUp(). High test coverage.

Grade: 8

Code readability:

Formatting:

- The method player.doUpdate has to many lines. This could be improved by splitting the method in smaller private methods. This also counts for the start method in the application class.
- The method doUpdate in the class levelEditor is to big. Split it in private methods.

Code is well formatted. makes good use of language conventions. excluding some long functions, overall the code is short and to the point. parts with different functionality are well divided.

Grade: 8

Naming:

- There are some magic numbers. In the enum ballmovement. These numbers could also be optimized. ese are also in the ball class. Improve this by creating static final variables.
- The test methods have many magic numbers. Try to create a variable for them with a good name.
- The test methods have numbers that are relative to each other. Try to make these into variables and make these variables relative to each other. This way you can see why those numbers are chosen to test with.

Ohter than magic numbers the naming is clear for a person looking at the code for the first time.

Grade: 7

Comments:

- The audioSystem class is missing javadoc.
- There is no javadoc in the test files. Someone who didn't implement the test doesn't know what they do. Thus can't improve/change them.

The test don't have javadoc. This was the only big problem for this category. The comments that were made had good explanations

Grade: 7

Meaningful enhancements to the other's group codebase.
The Code can be improved by implementing the model view controller design pattern thoroughly. This can be approved by removing the list of gameObject from screen and adding a observer pattern. If the state of the model changes the view should be updated. The advantages of a good mvc is that new screens can be added easily.

The method doUpdate() in the abstract class button is called each frame. But the subclasses that override the doUpdate() check whether the mouse is pressed. This can be improved by executing the code in the doUpdate method when the mouse is called with an observable. The buttons need to be registered to the mouseEvent or inputhandler.
This is the same for the doUpdate() method in the player, because for example when a player doesn't move because there's is no input why does it needs to be updated, so these are redundant calls.