Exercise 3 – Grading anon-group’s codebase

First impressions:

Sadly, we were not able to run the program because we were not provided with resources. Even after prying with the AssetManager it was difficult to pinpoint where the resource-folder was supposed to be located. As a first suggestion, it would be wise to provide your own exceptions to formalize the expectations you have! This way any other developer can immediately see what expectation he/she broke and localize the error more swiftly.

We found the following main gripes with your code-base:

1. **Amalgation of responsibilities:** There are classes that do too much. An example is BallManager (Almost 700 lines!). It appears to manage not only balls, but also seems to do hitboxing and game-states, managing the balls! This is too much; the linecount is indicative of this too.

   Point of action: Separate these responsibilities and delegate the checks, settings, etc to those components. Move hitbox-detection to a separate hitbox-manager. Move the update-logic for balls to the balls! Seriously! Every ball-entity should know itself what to do when it is updated and interact with the other components by itself. The ball-manager now acts as a god-class as it steers every other entity and contains the logic that should be part of the ball. If BallManager does 6 different things, that’s 6 different classes you can create to divide this problem into sub-problems!

   Consider that the `updateBalls` method is over 120 lines and contains 8, some even nested (nested) loops! This is way too much to understand and is unmaintainable. A single change can ruin everything and when multiple people work on this abomination it is extremely hard to track which changes to merge or not!

   Another example is the constructor of BallGraphBreadthFirstConditionalIterator. Too much in one thing!

```
public BallGraphBreadthFirstConditionalIterator(UndirectedGraph<Ball,
DefaultEdge> graph, Ball start) { …
```

   The constructor has multiple while-loops and many conditions! Why not separate this into methods that each do one or two things and then

2. **Overcomplicated methods**
   We encountered many methods that had a cyclomatic complexity (well) above 4 or contained many nested while-loops and if-statements. Please consider refactoring these big methods into smaller methods and, like point 1, make the responsibilities as atomic as possible! There were methods that used numbers where it was not clear what the method was doing. The functions are, in general, not self-documenting and this makes the code verry inaccessible to others to read, understand and maintain. The sparse javadoc does not make up for this! Don’t say what you achieve with a method, but explain what the method *is* doing. Explain the road *to* the result instead.

3. **Failing tests and lack of coverage**
   There were a few failing tests: SimpleScoreTest, boundaryTest and advancedScoreTest that failed because of a nullpointer-exception. This minnor thing aside, it was the coverage that caused the most concern. 32 tests only covered about 13% of the code-base. We think this is mostly because of your massive conditional statements. (E.g. BallManager).

Suggestions

- Introduce a Game-object that manages only the state of the game and initializes a Level.
- Leave level-initialization to a separate class btw(You already did this ☺)
- Give Ball-objects an update-method and let each Ball-object be updated ONE step when invoked.
- Move all responsibility of moving Ball-objects to the ball-object itself; e.g. a BallMoveBehaviour. The idea is that every update() call will call BallMoveBehaviour to step the movement in one direction.
- Similarly, move hitbox-checking to collision-component.
- Move AT LEAST every while loop in BallManager to a *separate* method. I don't care how you do this, but just move them out of their current place. This forces you to rethink *what* each part is doing and helps making your code more atomic.
- Give your assetManager a default-texture to return when nothing can be found, or document more clearly where it expects the resources to be located.
- Move the initialization of keys out of GameScreen. You can have a Controller for that.
- Most importantly, separate the functionality of Ball and BallManager properly! Make use of Strategy / Components to delegate responsibilities.

Grading

Source code quality: 5

You show some use of design patterns, but the massive methods destroy this rating, for abovementioned reasons.

Code readability 6

You have a lot of comments everywhere! This is a good thing, but sadly overshadowed by the giant methods ☹