Assignment 4

17 oktober 2016 - Group 666

Exercise 1 - Code Improvements

1: The switch that controls game states has been refactored. The game states are now separate classes (like the gumball machine example from the lectures).
2: Removed if-statements from wrap-around method in AbstractEntity by using modulo instead.
3: An attempt was made at refactoring the player's keyHandler method, but since this affected much of the other code in the end it turned out unfeasible to finish it before the deadline. The plan was to use a map<String, AbstractCommand> to put in a keyboard command and retrieve the class that describes the behaviour that should have happened.
4: Refactored if statement in setPlayerTwo method by using a ternary operator instead.

Asteroid onDeath method if statement could be refactored by using method overloading and different classes for the multiple types of asteroids.

Asteroid GetSurvivalSize method if statement could be avoided with a map that maps the asteroid radius to it's converted survival size.

Boss collide method could use method overloading instead of instanceof.

Exercise 2 - Teaming up

The exercise is complete all half build features. These features are: Audio, Highscores, Boss Battle and Survival Mode.

Requirements Audio

Must haves:

- The game shall include sounds.
- There shall be a background music.
- Shooting shall make a sound.
- Saucers shall make a sound.
- Boosting shall make a sound.
- Gaining a life shall make a sound.
- Hyperspace shall make a sound.

Should haves:

- All collisions shall make sounds.
- Player one and two shall have different sounds.

Could haves:

- The sound level shall be able to change in game.

Requirements Highscores

Must haves:

- The game shall be able to save multiple highscores in the same file.

Should haves:

- The highscores shall be sorted into the different modes.
- Names shall be stored together with the highscores.

Could haves:

- A separate menu shall be available for all highscores.
- Highscores shall be stored in a non-txt file (e.g. XML or json).

Requirements Boss Battle

Must haves:

- A special boss mode shall allow a player to fight a boss immediately.
- The boss shall shoot hostile bullets in the direction of the player.
- If the player touches the boss, the player shall die.
- The boss shall have multiple lives.
- The boss shall stay on the screen.

Should haves:

- In certain modes, the player shall be confronted with a boss after a period of time (for example, a certain amount of waves).
- If the player kills the boss they shall be rewarded with 20000 points.

Could haves:

- The game shall have different types of bosses that behave and look differently.

Requirements Survival Mode

Must haves:

- The game shall have a survival mode.
- Survival mode shall be available in coop.
- A player shall be able to chose these modes.
- Instead having waves, survival mode shall have a number of asteroids that will always be in the game.
- If the number of asteroids gets below the threshold, the spawner shall spawn more asteroids.
- The number of asteroids shall increase with the score.

Should haves:

- Highscores shall now have 4 different modes.
- Powerups shall be available in this mode.
- The logger will log all new actions correctly.

Exercise 3 - Walking in your TA's shoes

- Code quality*:
    - BubbleStorage appears to be a god class.
    - The number of classes seems rather high. I assume this is to split intelligence better and as a result make it easier to add more functionality? If so, good! Your code is very well-structured, making it easy to understand which parts of the code correspond to which parts of your application.
    - Lots of simple getters and setters that bloat your code and reduce readability. I suggest removing all getters and setters that lack special functionality and instead using Lombok or similar auto-getter/setter tools. Looking at your code, the number of lines in some classes will be cut in half, making it far easier to analyze and inspect your code.
    - You are using the depricated Date class in your Log class. Why?
    - No logging of exceptions?
    - Why use a vector for Player's observers? Wouldn't an ArrayList or LinkedList work just as well?
    - Powerup's apply method has a cyclomatic complexity of 12. Try to reduce this.
    - Bubble's get method returns null. Preferred would be to return an empty array so as to avoid null pointer exceptions.
    - Lots of code is copy-pasted for various screens. Inheritance is your friend here. Make use of it.
    - Your checkGridCollision uses != instead of ![object].equals([otherobject]). Why?

- Formatting/naming/comments
    - Javadoc for most methods. Nice! However, a lot of it appears to have been copied from one class to another without modifying it to fit.
        - Some of your javadoc is inaccurate, such as the description of pop() in Bubble stating that it sets the Bubble's state to POPPED. Remember, keep your javadoc up to date.
        - Some comments for your tests would be nice.
        - A lot of your javadoc is bare-bones, and not very helpful.
    - Strange use of whitespace. If you want to indicate separate portions of code, whitespace is fine, but in some classes I'm also seeing it between lines of code that are closely related and none of it between what appear to be separate portions of code.

- Building/tooling:
    - While you do not use PMD, activating it reveals a multitude of errors.
        - In particular, lots of use of extremely short variable names. In some cases, such as naming variables indicating the coordinates of something x and y,

this is understandable, but in other cases, such as naming a size-related variable n (BubbleStorage's method get(final int n)), not so much. In this example, it would be preferred if you used a more indicative name, stating clearly that that variable represents the size of something and what that something is.

- GameConfig and GameState are abstract, but seemingly only to prevent instantiation. They should be final classes with private constructors instead.

- I suggest using PMD and then spending some time cleaning various pieces of code or explaining why a given warning is not a valid complaint. Some of these issues are fairly serious (and impact the grading for code quality), but could easily be noticed and fixed if you used PMD.

- You lack findbugs. Using it results in findbugs reporting 174 bugs. You've got a lot of bugs to work on.

- Turning off checkstyle for specific lines of code is a bad practice.

- Testing:

- Since you do not have a separate package for your GUI-related elements, your line coverage appears to be only 35%. Looking at your code, you seem to have 8 major GUI-related classes, and approximately 80-170 lines of code, leaning to the higher end of that. Your other classes seem to have similar lengths, so, since there are so many non-GUI classes, this doesn't affect your coverage much. Test more.

- In particular, App has a rather strange test class. Three methods, none of which have @Test notations and none of which seem to implement a test?

- Branching/Code review:

- Without access to commit history or reviews, one cannot inspect the quality of the branching or said code reviews.

*Determining whether things that were changed were changed for the better is rather difficult to determine if one hasn't seen previous versions of the code and doesn't know exactly which features were requested.

Grading

- Code quality (25%): 5.5
- Formatting (3%): 7.0
- Naming (5%): 6.5
- Comments (5%): 7.5
- Building (5%): Not applicable.
- Testing (20%): 5.0
- Tooling (5%): 3.0
- Branching (10%): Not applicable
- Code review (22%): Not applicable
- Total (100%): 5.5

Next assignment:

A large portion of your code base has serious problems, and your tooling could use some work as well. Furthermore, your testing is rather lacking. As such, your next assignment is to implement the following checking tools: PMD and findbugs, and fixing the problems reported by these. Also, remove the lines that turn checkstyle off on singular lines of code and fix problems resulting from that. In cases where these are not possible, report why.