Assignment 5

24 oktober 2016 - Group 666

Exercise 1 - Anonymous peer suggestions

Responses to suggestions

1. A design pattern could be used for handling inputs, eg. observer pattern

Currently, we have a reasonably large number of design patterns (see list below for examples); adding more just for the sake of having more design patterns would run counter to the very purpose of design patterns, namely improving understandability (and being an easily implementable solution to a problem, although this is not relevant when one is implementing the design pattern to replace preexisting code).

1. Singleton logger
2. Abstract Entity class
3. Modes
4. States
5. Keyhandler

On a side note, in contrast to what is stated in the reasoning, we used the slides to make our implementation of a singleton-based logger, and as such, it *is* implemented according to the course's guidelines for singleton classes.

2. the Game class could be split into two subclasses of Game, SinglePlayerGame and MultiPlayerGame. This would make it easier to add extra game modes like a pvp mode.

We have effectively already implemented a split for game modes; however, we use Game for the basic functionalities, AbstracMode and its subclasses for the modes and Gamestate to store the information regarding modes and screen states. Effectively, since very little is actually different from mode to mode in terms of basic game logic, Game acts as the framework for the behind-the-scene workings and core mechanics, while GameState serves to hold the relevant information from AbstractMode subclasses, allowing us to easily add modes without having to go through the trouble of writing subclasses of Game for each new mode that we wish to implement.

3. Overall variable and method naming could improve, eg. ScoreCounter.startGame() doesn't start a game and Game.bullets() has seven method calls on the same line.

The reason ScoreCounter.startGame() was named the way it was is to indicate that it should be called at the start of a game. This is further clarified in the javadoc. Furthermore, Game.bullets() very specifically does *not* have seven calls on the same line. Instead, the calls are separated over multiple lines according to the functionality of the method calls, to allow for easy understanding of the use of each part of its functionality.

4. The interface could be clearer. I don't know what powerup is active when I picked it up. Also, options are non-existent. The only non-gameplay choice the player has is to launch a singleplayer game or a multiplayer game.

Icons for powerups were already a planned feature; as such, these have already been implemented by now. We did, however, add the option to mute sounds in response to your feedback. It might be worth noting that the original Asteroids lacked any such options, including choosing game modes.

Responses to reasoning

Regarding the use of Mockito and Cucumber: For the first several weeks of the project, attempts were made to get Mockito to work with our project. This proved fruitless and was abandoned as a result. As for Cucumber, the effort required to get it to work with our project was determined to be far too much for too little result; the consensus was that it would be better to spend said effort on improving the code quality.

One of the reasons we have relatively few non-javadoc comments is that we were told that our method naming was already very clear, to the point that the javadoc itself was practically redundant. As such, we thought the names of the methods called were sufficient indication of what was actually happening in the code.

Exercise 2 -

The initial InCode snapshot can be found on GitHub, in the Assignment5 folder directly on the project homepage.

For this exercise we had to pick the 3 worst design flaws (according to InCode) and fix them, or explain extensively why they shouldn't be fixed. Below, the three design flaws are explained, as well as how we fixed these flaws.

1. God class (Saucer)
The design error that was made while the saucer class was written is that the saucer class was given multiple responsibilities that can aren't very related, and additionally they are ones that can become pretty complex: shooting bullets, flying in different directions and maintaining information about the saucer object.

We decided to fix this by moving some of the duties of Saucer to a new class: SaucerShooter. This class would handle all the logic behind actually shooting bullets, and Saucer would only have to call the shoot() method of SaucerShooter without having to know any of the logic involved with shooting bullets.

2. God class (Player)
The design error that was made while the player class was written was similar to the saucer, the player was given more responsibilities than it should have been given.

Player being signified as a God class is something that can never be entirely fixed, as a lot of the information that Player uses that inCode sees as player using external methods are actually methods belonging to its superclass related to the location of the player (such as setX and getX), that are accessed a lot by the player class. This is functionality that doesn't belong to any type of entity but player.

Additionally, the test classes are seen as collaborator classes who intensively use Player's methods, which also contributes to player being seen as a god class.

That being said, we did solve some of the issues by moving the shoot logic out of the player class, as was done with the Saucer class. Additionally, some audio functionality was moved out of the class and the cyclomatic complexity of the methods was lowered.

3. Feature Envy (Startscreen method in StartScreenState class)
The startscreen method uses a lot of data from the GameState class. The design error that lead up to this is the GameState class, that originally used to contain information for all the different states and what to do in each state. As the amount of states grew, it was decided that the GameState class should be refactored and the states should become separate classes, and each state would contain the logic behind itself so that the GameState class would only have to concern itself with maintaining the states. However, this makes the states

and GameState heavily coupled, so there would be a point to putting the logic back into the GameState class.

This problem was solved by moving the startscreen method to the gamestate class, which is fine as the duties in the startscreen method are heavily related to the duties of the gamestate class (switching between screens). Now the startscreen method only has to make a single call, passing on the keyboard input to the GameState class instead of many.

Exercise 3 - Teaming up

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

Requirements

Must haves:

- The game shall have multiple bosses.
- The game shall spawn one of the bosses at random when a boss should be spawned (every 5 waves in arcade and every wave in boss mode).
- The Standard boss shall move randomly through the field, shall shoot 5 bullets per shot and shall take 10 shots (without powerups) to kill.
- The Teleport boss shall be a standard boss which teleports, instead of moving through the field.
- The Double boss shall be two standard bosses that shoot 3 bullets, are smaller and take only 7 shots to kill.
- Every boss shall have a fitting texture. The standard boss shall be an asteroid with an angry face. The teleport boss shall have a pulsating ring around itself. The double bosses will be normal bosses but have mirrored textures.
- All powerups shall have fitting texture.
- Powerups that are not yet picked up shall be stars with particles behind it.
- Health shall be a plus sign.
- Shield will be a circle (not filled in).
- Triple shot will be three dots (in this shape ^).
- Minigun shall be three dots (vertically stacked).
- The piercing bullets shall have an arrow through a broken line (->:).
- Large bullets shall have a filled in circle (smaller than shield).
- Highscores shall be loaded from a file in the correct order.

Should haves:

- The cursor shall not be in the way of the game.
- The player shall be able to navigate to the main menu through the pause screen.
- The main menu shall have the option to quit the game.

Could haves:

- Sounds shall not continue after death (player 2 booster sound).
- The game shall not have random lag spikes.