Assignment 4

TI2206 Software Engineering Methods (2015-2016 Q1)
Date: 16/10/2015
Group Number: 0
Group Members:

Thomas Baars - tcbaars@gmail.com
Lars Ysla - yslars@gmail.com
Boris Schrijver - boris@radialcontext.nl
Adriaan de Vos - adriaan.devos@gmail.com
Dylan Straub - d.r.straub@student.tudelft.nl

Table of Contents

Exercise 1 - Your wish is my command, reloaded

1. Requirements Power-up

The planned extension for our Fishy game is a movement speed boost power-up. The requirements for this extension are defined below. Each requirement for the planned extension is divided into subcategories based on their priority, where requirements marked “Crucial” are crucial for the extension’s implementation to be considered a success, and those marked “Non-crucial” are important but will only be implemented if time permits.

Crucial

- The game shall spawn a power-up every 2 minutes.
- The power-up shall have a unique sprite which will be displayed when spawned.
- The game shall remove the power-up when the player consumes it.
- The game shall start the power-up timer when the player consumes it.
- The power-up timer shall countdown from 10 seconds, after it has been consumed.
- Once the power-up timer has reached 0 it will expire, and the power-up will no longer be active.
- The game shall increase the movement speed by 50%, only if the power-up is active.
- The game shall change the movement speed back to the original speed, when the power-up is no longer active.

Non-Crucial

- The game shall spawn the power-up at a random location on the screen.
- The game shall despawn the power-up if it has not been picked up after 5 seconds.
- The power-up sprite shall be clearly visible with a bright yellow color.
- The game shall display a timer indicating the current time left for power-up, once consumed.

2. Analysis and Design

The analysis and design documentation produced based on the planned extension, of a movement speed boost power-up, are given below.

User Story

- As a player, I want the ability to consume a power-up which accelerates my movement speed by 50% when active.
- As a player, I want the power-up to last only 10 seconds before it becomes inactive.
- As a player, I want the power-up to spawn every 2 minutes.
- As a player, I want the power-up to be clearly visible.

Design

We plan on implementing it by mixing the Bubble and Enemy Entities and BubbleSpawner and EnemySpawner.

First we need to add the new “speedup” to all the current enums. So we need to add “speedup” to GameEntities, GameSprites and GameAnimations and add the right configuration options.

Then we need to create sprites/SpeedupSprite.java, entities/powerups/Speedup.java and entityspawner/SpeedupSpawner. This classes will be based on the current Bubble and Enemy classes. Most methods and code are partly the same so there will be a lot of inheritance and extending. Just like the current Bubble and Enemy classes we take a good look at what actions this object is responsible for and what information he needs to share.

We need to implement a way of tracking the current duration of the powerup. The collision/consuming between the player and the powerup can be handled just like the current enemys; the “consume(PowerUp powerUp)” method. If we handle it through the consume method it will be easy to adjust the movespeed through the Player class.

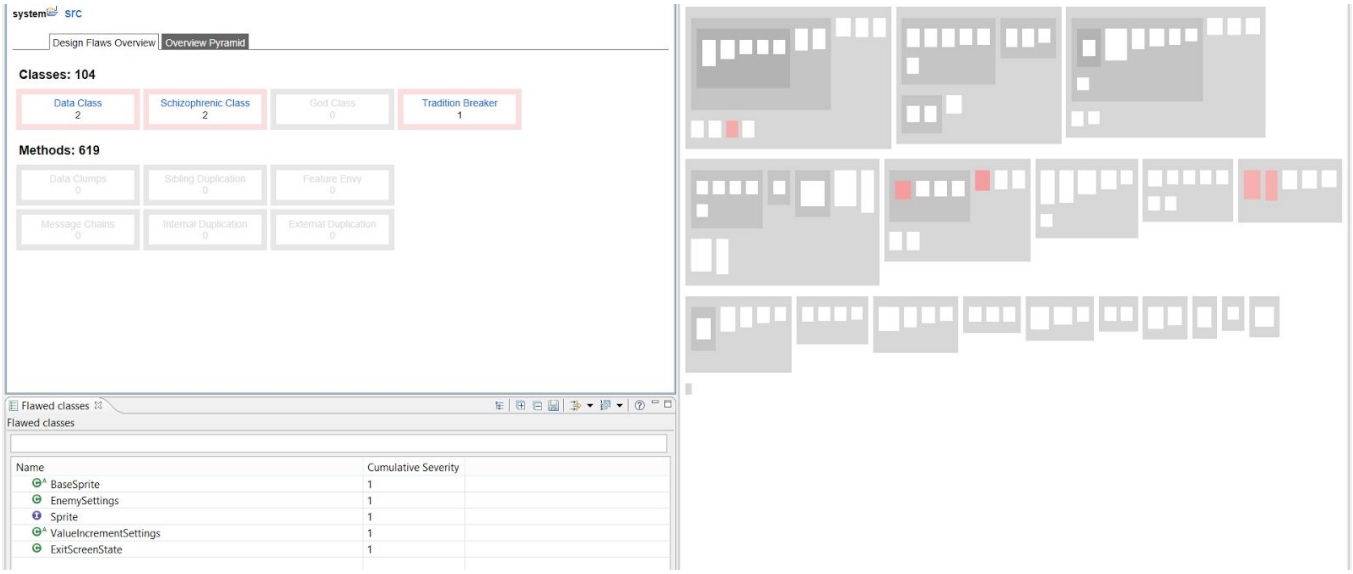
After this has been finished it is important that the classes/methods needs to be called by the rest of the software. That is quite easy with our current code layout, as this can all be done in the singleplayergames/ClassicGame.java class

This is our current idea of the implementation, changes can always occur during the development because we work iteratively. We will use pull-request code review with this implementation to ensure good and clean code.

Adriaan: Due to sickness, time constraints and big startup problems with my computer I did not have enough time to finish the implementation of this feature. I plan on finishing it this weekend. But that will be too late for the deadline.

Exercise 2 - Software Metrics

The result of using inCode to analyse the code quality of our source code, based on “Iteration 3”, is located in the file: *“src_1444722420924.result”* located at “\doc\Week 5 (12-10-2015 - 18-10-2015)” on the git repository. An overview of the result is located below:



The analysis revealed possible design flaws[1], which are described below:

Design Flaw I - Data Class

The “Data Class” describes a class which has an interface that exposes data members, instead of providing any substantial functionality. The classes mentioned in the resulting analysis file, which can be classified as “Data Classes”, are the EnemySettings class and the ValueIncrementSettings class. These classes expose a significant amount of data in its public interface.

The EnemySettings class has multiple attributes which represent the global enemy settings, such as the minimum number of enemies which can be spawned, the maximum number of enemies which can be spawned, and the rate at which they should be spawned. The class had methods to adjust these values which were being unused, so these methods have been removed. The removal of these methods meant that the values could be made constant, which restricted the impact of accessing the data members through the public methods. However the spawn rate has not been set constant as we may want to change this to change the difficulty in later versions.

[1] Design Flaws - http://www.intooitus.com/products/incode/detected-flaws

The ValueIncrementSettings class has multiple attributes which represent the average values of the available entities, such as the average minimum area, the average area, and the average maximum area. Upon inspecting the class and its subclasses, it was decided that the hierarchy only existed to prevent code reuse. So the hierarchy was dissolved and the attributes were moved to a specific class: AverageEntityValues. The new AverageEntityValues class still suffers from external usage of public data.

We have multiple classes which could also suffer from external usage of public data but this is not a severe design flaw; but rather our way of allowing the game's dynamic global settings among multiple classes. The global settings are stored as attributes and accessed through public methods in a controlled way.

Design Flaw II - Schizophrenic Class

The "Schizophrenic Class" describes a class with a large and non-cohesive interface. Where non-cohesive means several disjoint sets of public methods that are used by disjoint sets of client classes. The classes mentioned in the resulting analysis file, which can be classified as "Schizophrenic Classes", are the BaseSprite class and the Sprite interface.

The Sprite interface defined multiple public methods which were being unused or only used locally. For example the getSpriteX, getSpriteY(), getSpriteWidht(), and getSpriteHeight() methods. These methods have been removed.

The BaseSprite class, as a by-product of a non-cohesive interface, is non-cohesive; as well as defining its own methods which were only used locally. For example the getFrameWidth() and getFrameHeight() methods. Visibility of these methods have been changed to more appropriate values, and the class and its subclasses have been organised into a more cohesive way.

The size of the Sprite interface and the BaseSprite class has been reduced, and is now more cohesive.

Design Flaw III - Tradition Breaker

The "Tradition Breaker" describes a class which breaks the interface inherited from a base class or an interface. The class mentioned in the resulting analysis file, which can be classified as a "Tradition Breaker" is the ExitScreenState class.

The ExitScreenState class has multiple methods which override the parent class but have no implementation. For example the handleKeyPressed() and handleKeyTyped() methods. The

methods have no implementation because they are not applicable; however the unused methods have been removed for better organisation. The methods may be implemented in a later version, if we decided to display a confirmation screen before exiting the application.