SOFTWARE METRICS IN SPACE INVADERS

SPRINT 4

by

A. Smesseim, S. Baraç, A. el Khalki, A.W.L. Oostmeyer, M. Maton

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at the Delft University of Technology,

to be presented on 16 October 2015.

Supervisor: Dr. A. Bacchelli

TU Delft Delft University of Technology

CONTENTS

1

YOUR WISH IS MY COMMAND, RELOADED

1.1. REQUIREMENTS

1.1.1. FUNCTIONAL REQUIREMENTS

For the game Space Invaders, the requirements regarding functionality and service of the saving and loading functionality are grouped under the Functional Requirements. Within these functional requirements, four categories can be identified using the MoSCoW[1] model for prioritizing requirements.

MUST HAVES

1. The game must show a "Save game." button when the user is in the store.
2. When the player presses this button, the number of lives, bombs, points, the current level and the intact (non-destroyed) barricades are written to the file named "spaceinvaders.sgfile". The text of the button changes to "Save game. Done!".
3. The game must show a "LOAD GAME" button in the main menu.
4. If the player presses on this "LOAD GAME" button, and the save file "spaceinvaders.sgfile" is available, then game shows the store, with the exact same values for the number of points, bombs, lives and the current level as the last time the player pressed "Save game.". If the player then presses "Continue", then the game shows the same barricades as at the end of the level before "Save game." was pressed.

SHOULD HAVES

5. If the player presses on this "LOAD GAME" button, and the save file "spaceinvaders.sgfile" is available, and then presses "Continue", then the game shows the same barricades as at the end of the level before "Save game." was pressed.

COULD HAVES

6. If the player presses on this "LOAD GAME" button, and the save file "spaceinvaders.sgfile" is not available, then the text "LOAD GAME" changes to "LOAD GAME (failed).

WOULD/WON'T HAVES

7. The game won't support saving the player's progress to a different file than "spaceinvaders.sgfile".

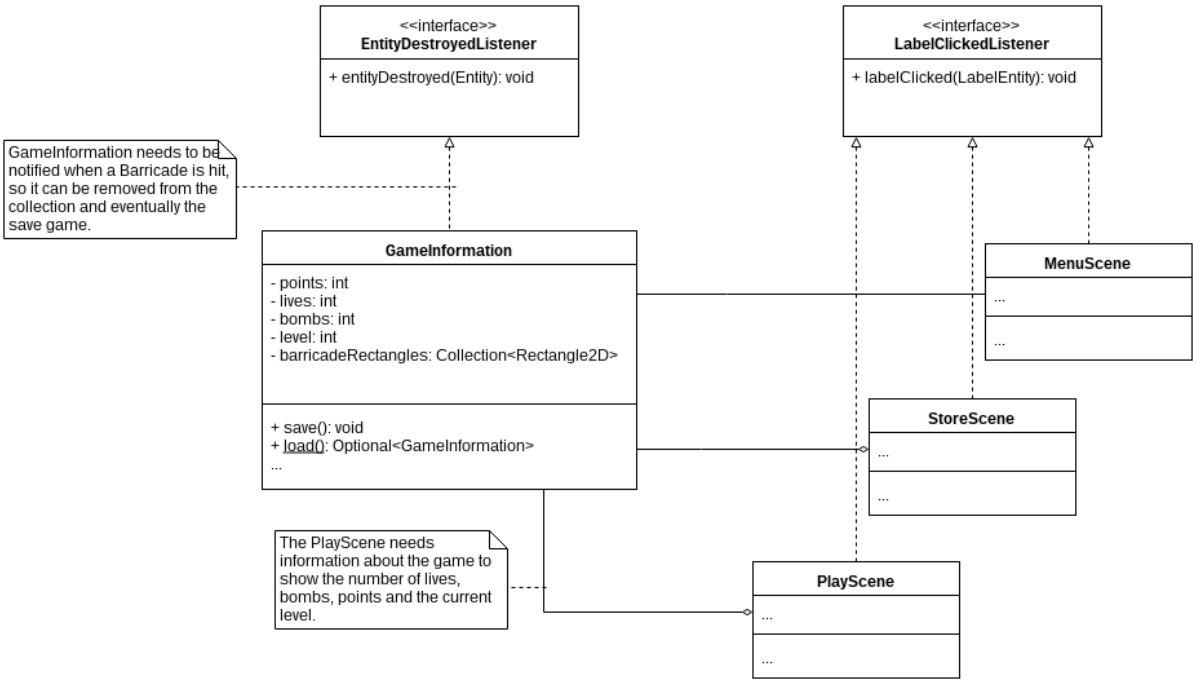
1.1.2. NON-FUNCTIONAL REQUIREMENTS

In addition to the functional requirements specified in chapter 1.1.1, the game should also adhere to several non-functional requirements, outlined in this chapter. These requirements will not affect the functionality of the game, but will ensure that the game is gradable by the instructors of the course TI2206. The non-functional requirements of the game are:

8. The game must be playable on Windows (7 or higher), Mac OS X (10.8 and higher) and Linux.
9. The game must be implemented in Java.

10. Scrum methodology must be applied.
11. The source code must be hosted on GitHub as a public repository.
12. The version control used for developing this game must be Git.
13. The tests must be automated using JUnit.
14. This project must use Maven for build automation.
15. This project must use Travis CI for continuous integration.
16. This project must employ the following static analysis tools: Checkstyle, PMD and FindBugs.

1.2. UML



1.3. CRC CARDS

GameInformation	
Subclasses:	None
Superclasses:	None
Responsibilities:	Collaborators:
Storing the bombs, lives, points and barricades Saving the game Loading the game	Collection & Barricade OutputStream InputStream

StoreScene	
Subclasses:	None
Superclasses:	GameScene
Responsibilities:	Collaborators:
Showing labels and performing right action upon click Make next level Switch scene to next level Initiate saving	LabelEntity PlayScene Engine GameInformation

PlayScene	
Subclasses:	None
Superclasses:	GameScene
Responsibilities:	Collaborators:
Creating game entities	Entity
Creating the store	StoreScene
Switch scene to the store	Engine
Displaying the lives, bombs and points	GameInformation

MenuScene	
Subclasses:	None
Superclasses:	GameScene
Responsibilities:	Collaborators:
Showing labels and performing right action upon click	LabelEntity
Creating the playing field	PlayScene
Switch scene to the play scene	Engine
Initiate loading	GameInformation

2

Software Metrics

2.1. inCode analysis file

The inCode analysis file is located at deliverables/spaceinvaders18_1444722105720.result.

2.2. Design flaws explanations

2.2.1. Data clumps

The first design flaw inCode found was the group of combined constructor parameters we're using everywhere: positionX, positionY, width, height. It indicated that those parameters could be good candidates for grouping into an object. The Entity class was fixed and with that all inheriting classes were fixed as well.

2.2.2. Data class

The remaining code smell inCode detected was a Data Class named DummyColliable. This class is used to allow non-entities to have a one off collision with entities in the game. Examples of this are handling mouseclicks or determining what items are in the blast radius of the bomb. It is the simplest implementation of the Collidable interface and is read only.

It is not desirable to move collision code into the DummyColliable class as that would require adding the same method to all other classes implementing Collidable. Collidable cannot be converted to a (virtual) class as we want our classes to inherit from Entity, which gives a better class model. Following this reasoning we decided not to remove or change the DummyColliable class.

2.2.3. God class

As we didn't have any code smells left after fixing the first one and disregarding the second we are left with describing a flaw that inCode could pick up but wasn't present in our design. For this we chose the God Pattern. Our initial design contained one class, GUI, that handled all logic. After the first week we slowly moved most logic from there to individual Entity classes. This ensured easier testability and made it easier to work together on the same code base without interfering with each other.

BIBLIOGRAPHY

[1] S. Ash, *Moscow prioritisation briefing paper,* DSDM Consortium (2007).