Assignment 4

Game: Temple Run

by

Group 9

Course: TI2206

Software Engineering Methods

Maarten Sijm
Robin van Heukelum
Mathias Meuleman
Mitchell Dingjan
Maikel Kerkhof

TA: Jurgen van Schagen

Teacher: Alberto Bacchelli

16-10-2015

Exercise 1 – Your wish is my command, Reloaded

Our TA has given us the assignment to make a Highscore server, on which we can send the scores of the players.

Requirements

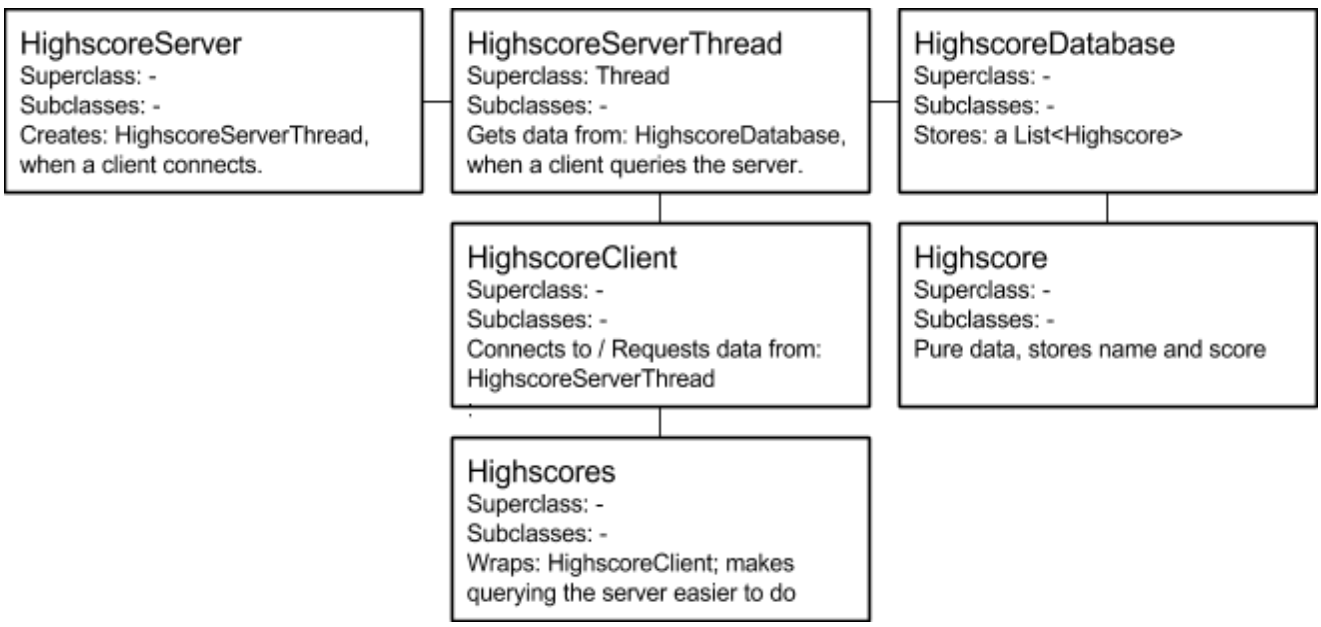
Requirements for the game:

- The game shall feature a highscore functionality.
    - Every time the player dies, the score gets sent to an external server.
    - The user shall be able to view its ten best scores ever achieved.
    - The user shall be able to view the ten global highscores.
        - The global highscore list shall only contain the best entry per user.

Requirements for the server:

- The server shall be able to answer correctly to queries that the client sends.
    - When the client's query is "add <name:string> <score:int>", a new highscore shall be added.
    - … "get global <amount:int>", the server shall send back a list of global highscores, length *amount.*
    - … "get user <name:string> <amount:int>", the server shall send back a list of user highscores, scored by *name*, length *amount.*
    - … "exit", the server shall close the connection with the client.

CRC cards



Brief overview of the working of our HighscoreServer

The HighscoreServer has a main method, meaning that the server can and should be run separately from the rest of the Application. It has a ServerSocket listening on port 42042, and whenever there is an incoming conneciton, a new HighscoreServerThread is spawned.
This Thread listens for input from the client, which can be any query. The query will be passed to the HighscoreDatabase, which will return a String that can be sent back to the Client.
When the Client receives data, a callback function will be called. This callback function has to be defined by the user. The String can be parsed and turned into a success flag or a List<Highscore>, which is done by the

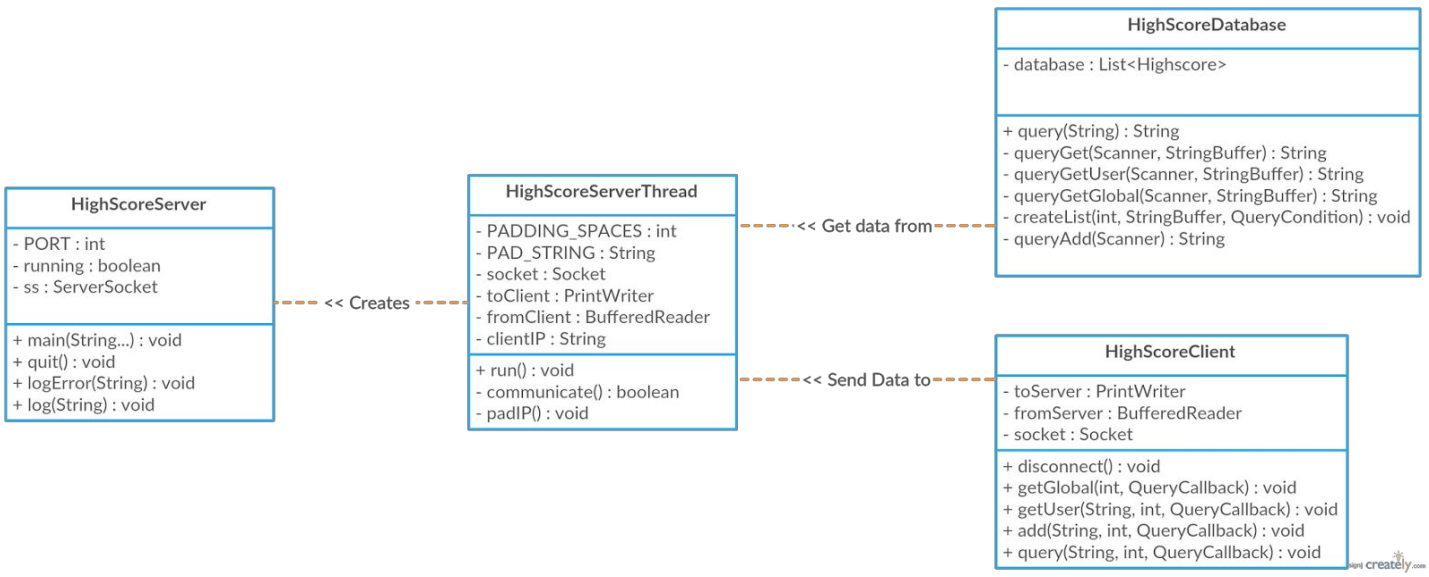
Highscores class. Data can be retrieved easily via this class, which makes it easier to implement it in the front-end.

Responsibility Driven Design

We made sure that every class has exactly one responsibility.

- The Server listens for incoming Client connections, as a Server ought to do.
- The ServerThread handles the communication with one particular Client.
- The Database stores the Highscores and can be queried to add/retrieve highscores.
- The Client is on the other end of the connection, communicating with the Server.
- The Highscores class wraps the Client to make the communication easier.
- The Highscore class purely stores data, and has possibilities to parse Strings into Highscore objects.

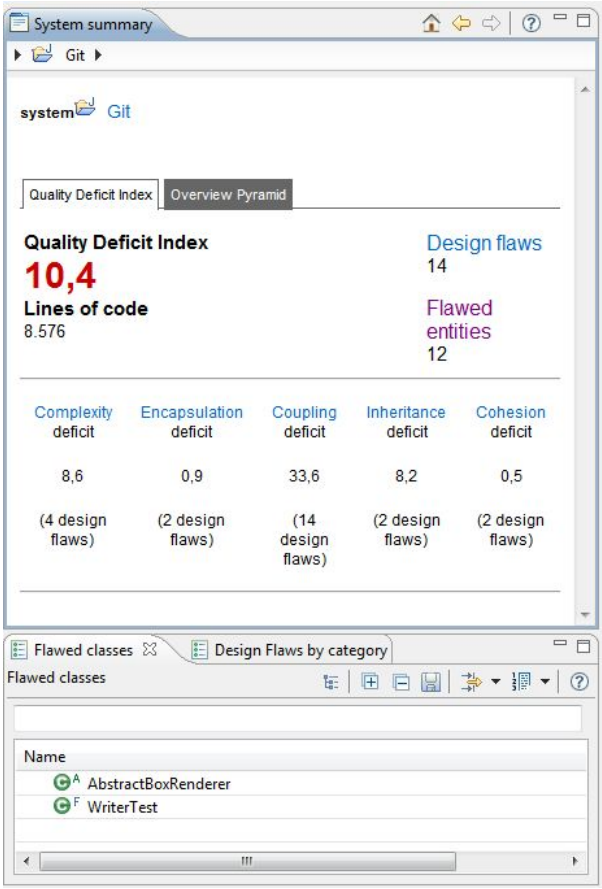
UML Class Diagram



Exercise 2 – Software Metrics

1. Use inCode to compute software metrics on your project

Our result is located in our Git repository at `/doc/inFusion/Git_1445021857971.result`.



2. Analyse and (if possible) fix three design flaws

SAP Breakers: 2	
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\gui\skin	8
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\level	6
Cyclic Dependencies: 10	
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\level	2
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\gui\skin	2
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\level\save	2
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\util	2
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\gui	1
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\gui\renderer	1
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9	1
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\gui\scene	1
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\level\entity	1
subsystem_C:\Users\Maarten\Dropbox\School\TI2206-Software Engineering Methods\Git\src\main\java\nl\tudelft\ti2206\group9\audio	1
Feature Envy: 2	
updatePosition() : void	2
testSaveGame() : void	2

We picked the following three design flaws:

1. **SAP breakers**
2. **Cyclic dependency**
3. **Feature Envy**

a) Explain the design choices or errors leading to the detected design flaw.

1. The skin and level packages are apparently extensively used in other packages.
2. In our project there were 10 cyclic dependencies noticed by inCode. These 10 cyclic dependencies had mostly to do with classes a couple of classes that were dependent of each other.
3. There are two methods that show Feature Envy:
    a. In AbstractBoxRenderer.updatePosition(), the points of the tracked Entity are used extensively.
    b. In Writer.testSaveGame(), lots of State properties are set in order to test the saving funtionality.

b) Fix the design flaw or extensively and precisely explain why this detected flaw is not an error and, thus, should not be fixed.

1. Both heavy dependencies cannot be fixed.
    a. The “skin” package is depended on by many other packages, including “gui.renderer”, “level.save”, “gui”, etc. However, the functionality that is needed, belongs to the “skin” package, and upon moving this functionality, there would be responsibility problems. Therefore, we decided not to fix this flaw.
    b. The “level” package is also depended on by many other packages. For example the InternalTicker and Track are widely used. These classes represent the core of the system, and we tried to fix some of these dependencies (see 2a of this assignment). However, inFusion still displays the warning. We decided not to continue moving dependencies away from the “level” package, again due to responsibility problems.
2. For some of the 10 cyclic dependencies we decided to break the cycle, for some we didn’t. We have moved some dependencies to the Observer pattern in our game.
    a. One example: in InternalTicker.step(), when the Player died, the Ticker would invoke some methods in GameScene to stop the game. Now, GameScene has a GameObserver, which observes when the player dies, and stops the game when this happens. This removes the dependency of InternalTicker on GameScene.
    b. For the others, there is a good reason not to remove the cyclic dependencies, take the packages “gui” and “gui.scene” for example. ExternalTicker puts things in the GameScene, and GameScene starts and stops the ExternalTicker.
3. We decided to fix updatePosition(), and let testSaveGame() be.
    a. We fixed updatePosition() by setting the center and size of the tracked entity as fields in the Renderer. Now we have to call entity.getCenter() and entity.getSize() only once, instead of three times per method per tick.
    b. As the test *should* test in many different areas of the code, we decided to leave this test be. In order to test the SaveGame functionality correctly, we have to unset any saved variable from their defaults.