Peer Review

By group **16**[1] to group **13**

In the Peer Review we try to give examples of improvement and solutions.

Do the design and implementation follow design principles?

One example of code reusability is that there exists an inheritance hierarchy to be able to reuse code. Even if there is an opportunity to reuse code, there still is duplicated code. For example in all of the subclasses to Combatant, there are getters for Hp, Atk, AtkRange and also idecHp() and incHp(). These methods already exists in Combatant and are unnecessary to have in the subclasses.

The code has a lot of small classes which is positive due to the fact that Single Responsibility Principle is used and therefore the structure is more modular and easier to maintain.

There are some methods in the code that are way too big (for example see World Class and View Class) and should be separated according to Separation of Concern.

The constructor of the class World is very big and does a lot of things. A solution could be to create methods that does the same things but are placed outside of the constructor and instead are called when needed to be used in the construction.

As the group discuss in their system design document (SDD) they use 4 obvious design patterns. As a system architecture design pattern they use a standard MVC pattern which seems to be a fair idea as they're building a game with a heavy visual representation.

In terms of Creational Design Patterns they seem to take advantage of two types. They use a Singleton Pattern at the creation of the "OpenSimplexAdapter" and "ItemFactory". This is a good idea as multiple instantiations of these classes may and probably will cause problems. This design pattern could be used more as it seems like multiple classes should only be created once. The other Creational Design Patterns that they use is a Factory Pattern in the form of the "ItemFactory" class. Studying the code and UML diagram this factory only seems to be responsible for generating random items (maybe used in some parts of the game). This creational pattern could be used in more parts of the game to create objects, enemies, items etc.

The group makes an attempt to decrease the coupling in the application by a Facade Pattern for the model. Its functionality could however be expanded to make it even better.

Is the code documented?

There is no JavaDoc in all of the code. There are however some comments in the code (probably to help themselves understanding what a certain test or method is doing).

Are proper names used?

There are some methods with the name Hp. Hp could be a bit confusing and should be rewritten to make its purpose more clear.

There is a class called model, which is confusing due to the fact that they are using the MVC structure. Model is the name for the package (in the MCV structure) that holds all of the logic and data and should not be used as a class name. An improvement could be to rename the class so its name represents the game due to the fact that it is the codes facade-class.

Is the design modular? Are there any unnecessary dependencies?

Another example of lack of code reuse and missed opportunity to reap the positive aspects of inheritance is the following:

[1] Elin Nilsson, Tina Samimian, Jacob Messinger, Oscar Helgesson, Norbert Laszlo

World is depending on Entity, Movable, Combatant, Enemy and Player. Combatant is a subclass to Enemy and implements Movable. Due to the fact that the positive aspects of inheritance is not used to the max, there are some unnecessary dependencies.

Does the code use proper abstractions?

As the current structure suggest the abstraction for the application works fine due to the fact that they use proper inheritance hierarchy and try to create abstractions for all classes.

Is the code well tested?

There are only four classes that have 100% method coverage. Many classes in the Model-package are tested but not fully. A test package should reflect all of the other classes like a mirror. This means that in a perfect scenario each class should have its own test class.

The test-methods do not have describable names which leads to difficulty to read and understand what the methods actually are testing.

With this said, the test-methods should be testing more of the logic and should be split up into smaller tests that only tests one scenario.

Are there any security problems, are there any performance issues?

As written in the SDD, a lot of class members do not have the correct access modifier (private, protected, public and package-private) and expose too much information and behaviour.

Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?

The code is relatively easy to understand due to the fact that many of the classes are small and therefore easy to read and analyze. However there are some things that could be done in order to make it easier to understand. For example better names to variables, methods and classes and also better documentation.

The packages are separated from each other in a way that follows the MVC structure.

RAD-Comments

There appears to be a lack of User Stories that explains basic logic of the program. Listed below are some examples:

- Create a User Story to further explain that you should not be able to move through obstacles. Currently none of the User Stories explains this fact, but it should still be a vital part of the final product.
- Create a User Story that explains that the character should be able to die.
- Create a User Story that explains when sound should be played (Unclear if by Noise in the code, you mean noise as in playing a noise, or if noise is something that is affected by some outside variable (See swedish “brus” or “störning”)

Should probably make it clear whether or not a User Story is an Epic or User Story. I.e. the first User Story is basically just an epic explaining what the end goal of the application will be. This will also make it easier to distinguish between what the intended purpose of each User Story is.

Also don’t forget to update what is implemented and what is not. Right now it says that STK003 is not implemented, although it seems to be implemented already.

STK003 should be re-written to be “As a User” instead of “As a Developer” as it is not of importance to have this User Story if it is not a demand from the costumer.