

<i>David Alderliesten</i>	4368703
<i>Jesse Tilro</i>	4368142
<i>Jeroen Meijer</i>	4382498
<i>Floris Doolaard</i>	4362748
<i>Niels Warnars</i>	4372069

4368703

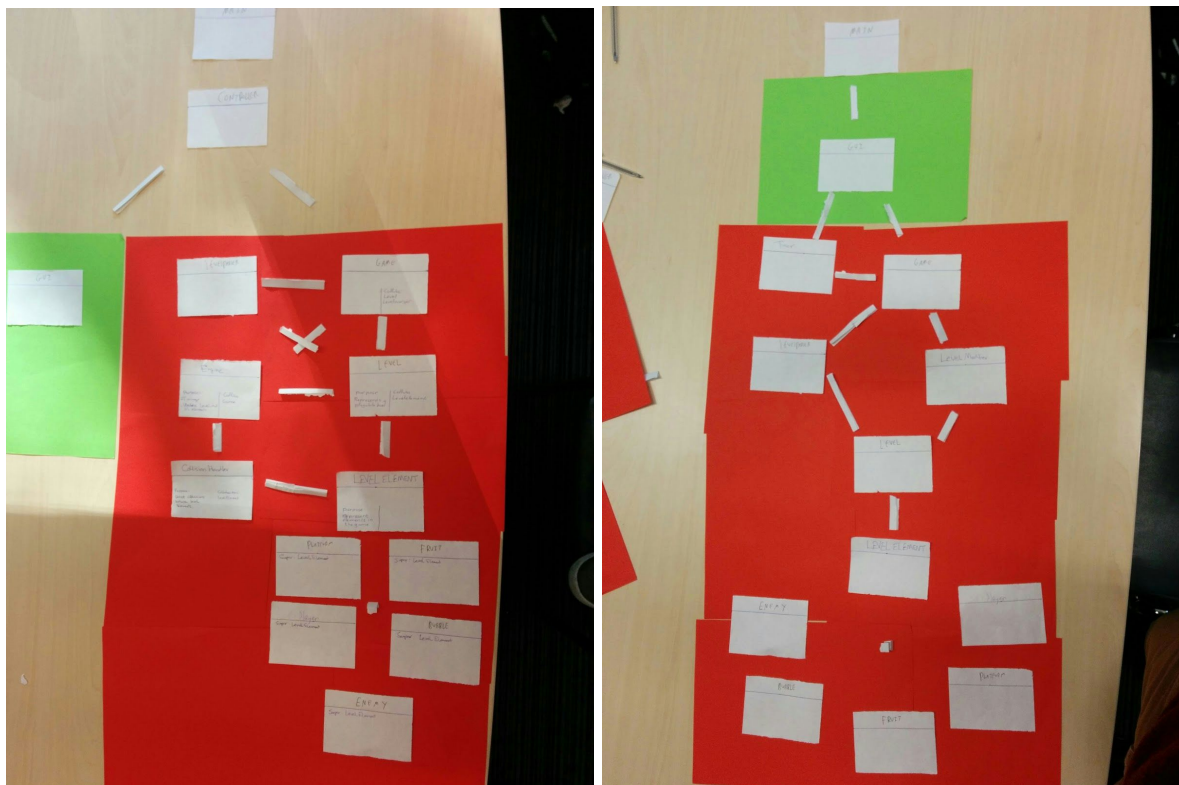
4368142

4382498

4362748

4372069

## 1.1 Responsibility Driven Design



## 1.2 Main Classes

### Controller

Collaborators:	Purposes:
Game	Connects the model (Game) and the view (GUI).

### Game

Collaborators:	Purposes:
LevelParser	Represent a game with all related attributes.
Level	
Engine	

### Engine

Collaborators:	Purposes:
CollisionHandler	Apply physics and other events to a Level.
Level	

### Level

Collaborators:	Purposes:
LevelElement	Represent a Level in a game.

### LevelElement

Collaborators:	Purposes:
	Represent all the elements in a given Level.

### GUI

Collaborators:	Purposes:
----------------	-----------

	Visualise a game in its current state.
--	--

### **LevelParser**

Collaborators:	Purposes:
	Loads a file from a text file.

### **CollisionHandler**

Collaborators:	Purposes:
	Detects collisions in a Level.

## 1.3 Less important Classes

The Classes not mentioned above are the subclasses of LevelElement and they did not seem significant enough to specifically mention as they closely resemble LevelElement.

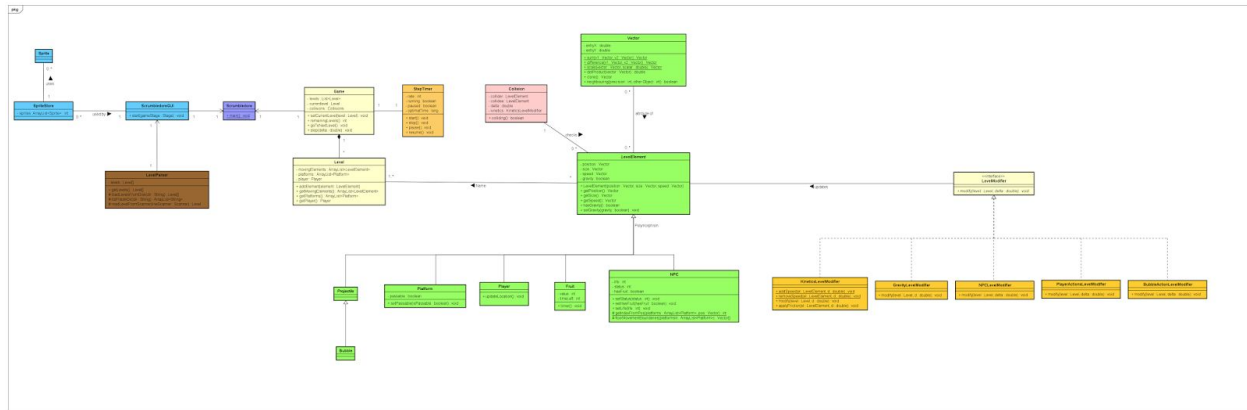
As for the differences between our setup and the newly derived one, they are as follows:

1. The Engine class is split up into the StepTimer and various LevelModifiers. If we put them all in one Class it would have been a mess now it is much more readable.
2. The missing controller, because of JavaFX demanding to be the main Class that has to be run we could not implement a separate controller class.

On all other accounts the two models resemble each other very closely, as such we did not change anything to our codebase.

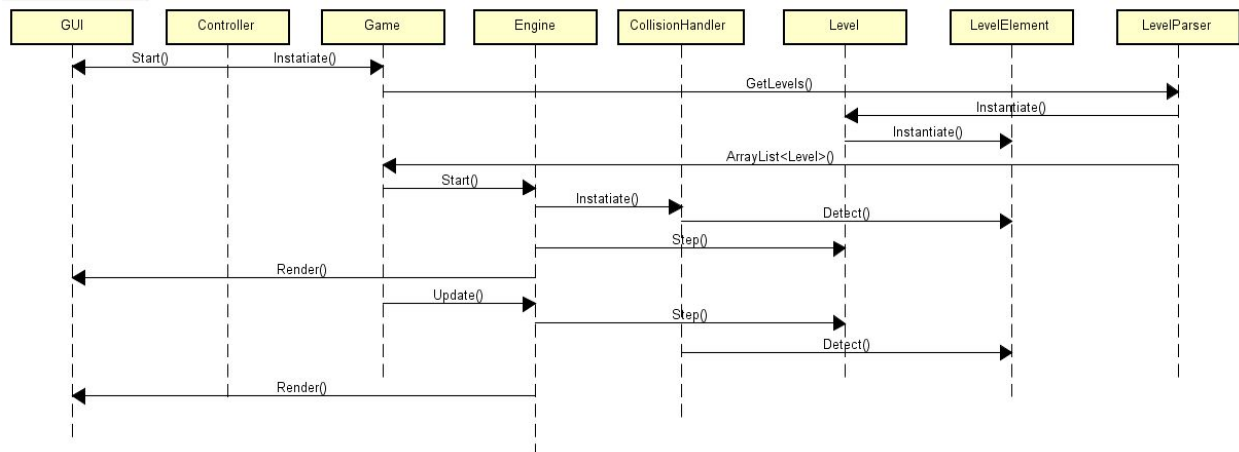
## 1.4 Class Diagram

The raw version of the class diagram ([Scumbledore 2.1.asta](#)) can be found in the Git repository under the “documents” folder.



## 1.5 Sequence Diagram

The raw version of the sequence diagram ([Bubble\\_Bobble\\_Diagram.asta](#)) can be found in the Git repository under the “documents” folder.



## Exercise 2 - UML in practice

1. If the relation between two classes is a composition it would mean that one class is a definite part of the whole, which is the other class. On the other hand, an aggregation relationship would mean that the class could be a part of the other class, but could also be independently used.

In our own game for instance, we implemented the classes Game and Level. Since our game consists of level and can't be played without them, the relationship between Game and Level can be called a composition.

2. Currently we do not have any parameterized classes. You can use these sort of classes if these classes only use certain parameter types. For instance, a list that only contains

instances of the type String and only has methods that process this certain type, you can parameterize this class.

3. We have one 'polymorphism'. We used polymorphism for the subclasses of LevelElement so that we could make a lot of different level elements in the game. According to the lecture none of our current hierarchies should be removed since no 'reuse'-hierarchy has been used.