TI2206 Software Engineering
Bubble Shooter report: Assignment 1
EEMCS/EWI

Gerlof Fokkema 4257286
Owen Huang 4317459
Adam Iqbal 4293568
Nando Kartoredjo 4271378
Skip Lentz 4334051

September 21, 2014

Exercise 1: The core

In this section the exercises 1.1 up to 1.4 are elaborated with fully in depth explanations.

Sub-exercise 1.1

The two **main** classes that have already been implemented, described in terms of responsibility and collaborations are as follows:

1. First, one of the most important classes is the *Board* class. This class describes the entire playing-field. It can also place and remove bubbles, and it can return the clusters of bubbles of the same color. It collaborates with the *BubbleShooterScreen* class and the *Bubble* class.

2. Secondly, the *BubbleShooterScreen* class should also be considered one of the main classes. It is responsible for drawing all the game elements (bubbles, cannon and background) onto the screen, for the game logic and for applying the game's rules. Additionally it takes care of the user input and acts accordingly. It collaborates with the *Board* class and the *Cannon* class.

Sub-exercise 1.2

The reason why the other classes are not considered main classes, is because of the fact that they carry far less responsibility. Those classes do not collaborate with that many classes in comparison to the main classes. For example, the *Bubble* class does not "know" about any other classes, and its only responsibility is basically to exist.

Looking at the current structure, some changes to the non-main classes are in order.

First up are the *BackgroundMusic* and *SoundEffect* classes. These two classes both take care of the audio of the application. While it is not necessarily bad to have these two seperate, the responsibility these two classes have are somewhat similar; one manages the background music and the other the various sound effects. As previously mentioned, this simply relates to the audio. So these classes can be easily merged together into one larger class: the *SoundEngine*.

The next issue, that should be addressed, is loading assets into the game. At first this was just done in every single class where the asset was necessary itself. This caused everything to become rather cluttered, so it would make sense to do all the asset loading in another place. This resulted into introducing a new class: the *Assets* class. All the assets would then be loaded from the start of the application in the *Launch* class.

Sub-exercise 1.3

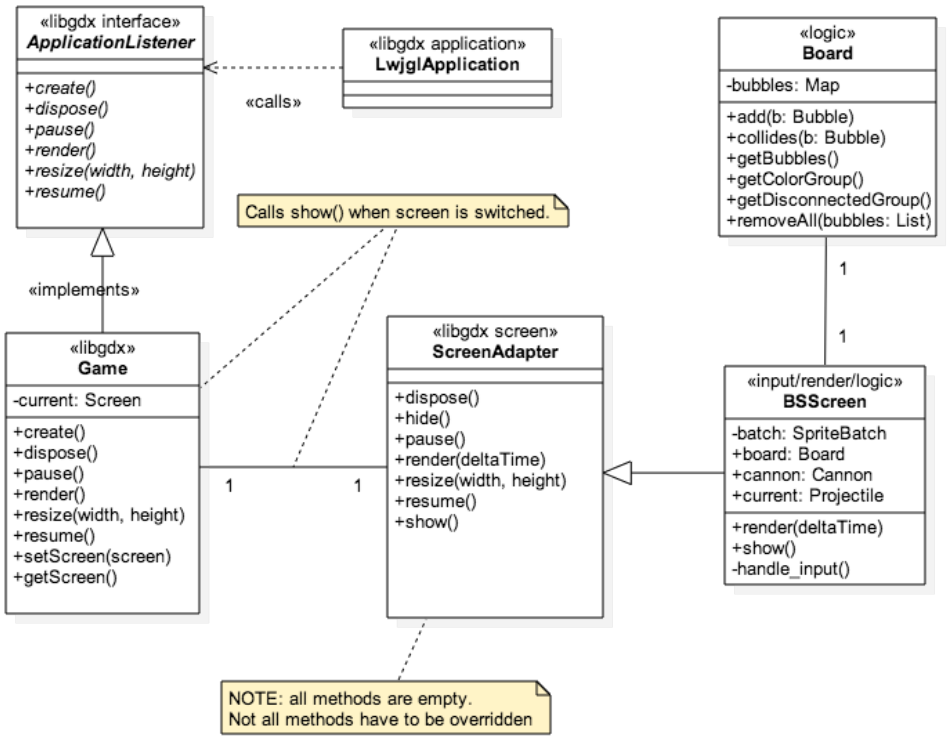

Figure 1: The class diagram after the first sprint

Sub-exercise 1.4

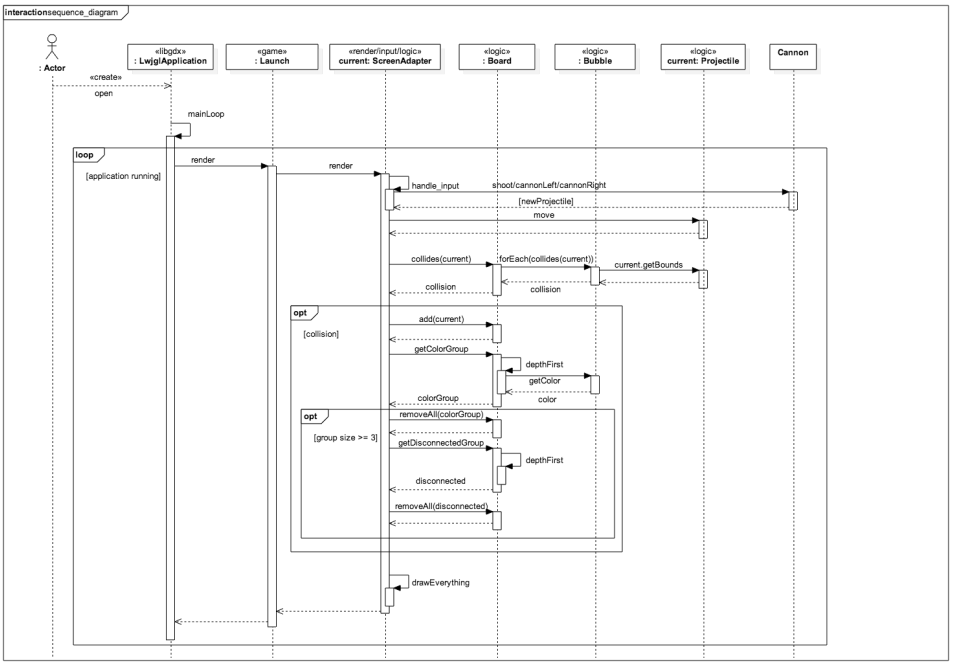

Figure 2: The sequence diagram after the first sprint

Exercise 2: Theory in practice

In this section the exercises 2.1 up to 2.3 are elaborated with fully in depth explanations.

Sub-exercise 2.1

1. **Aggregation** is a special form of **association**. This is best described as a 'has-a' relationship between a parent and the child, meaining that the child does not necessarily need the parent in order to simply exist.

2. **Composition** is a special form of **aggregation**. This differs from aggregation, in the following: the relationship between the parent and child will not hold when the parent does no longer exist. In other words, the child may not exist without its parent.

Inside the project, there are two cases of **composition** to be found.

Firstly, the *Bubble* class has such a relationship with the *Board* class. The *Board* contains lots of bubbles, so when the board stops existing so do all the bubbles.

Secondly, the various classes that adapt the *ScreenAdapter* class, such as the *MainMenuScreen*, have a similar relationship with the *Button* class. When the main menu vanishes, the buttons follow.

Furthermore, several cases of aggregation also occur. However these relationships currently exist, they are not favorable and should be changed later on (because it does not hold a 'logical connection' with the corresponding classes). These relationships can be found inside the *BubbleShooterScreen* class. The *BubbleShooterScreen* currently holds three *aggregation* relationships with the *Cannon*, *Board* and *Projectile* classes. This is due to the fact that the *BubbleShooterScreen* contains these elements, however they do not necessarily need the *BubbleShooterScreen* in order to exist.

Sub-exercise 2.2

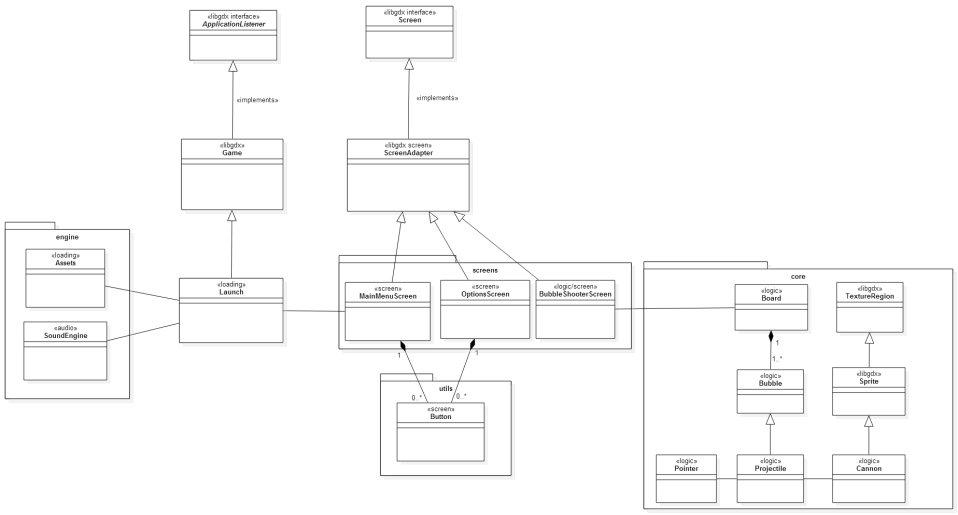

Figure 3: The class hierarchy at the current state

The following class hierarchy was made without a lot of careful thinking beforehand. So it was mostly done on a whim. The various kinds of relationships are also marked in the diagram with different symbols. The black diamond is a composition relationship, the arrow is a generalization (which indicates subclassing). Furthermore the regular line is a association. The *Board* and *Bubble* relationship was also explained in **Sub-Exercise 2.1**, this also goes for the *Screen* classes (including the *Button*).

Sub-exercise 2.3

There are cases of if-statements to be found in the project. Most if-statements from the *BubbleShooterScreen* have been refactored. A new structure for handling user input has been introduced to reduce the amount of if-statements: these include the *AbstractProcessor* and the *SinglePlayerProcessor* classes.

Instead of checking every time whether a given key has been pressed, all the keybindings are put into a Map. Only a single lookup into the map has to be done, in order to check if there has to be some kind of response to the user input.

Other cases where if-statements were used, should remain. This is because there is no valid reason to replace them, because there is no profit to gain from doing so. Additionally these if-statements are not if-statements, that are 'out of place'. They provide structure to the code and cannot be replaced by f.e: a polymorphic structure.

Exercise 3: Play with your friends

Sub-exercise 3.1

The implementation can be found in the attached files.

Sub-exercise 3.2

The documents derived during this phase can also be found in the same place as previously mentioned. This is a snapshot moment.

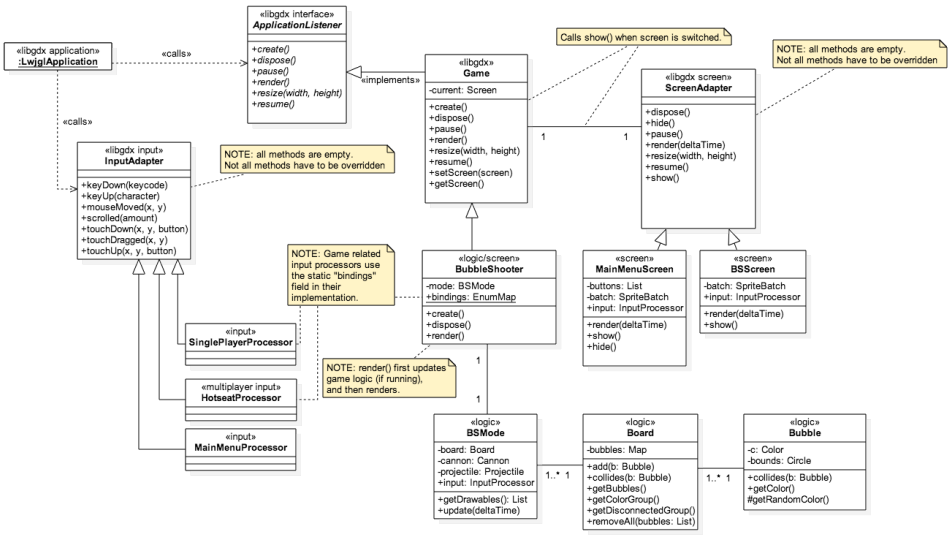

Figure 4: A snapshot moment from the new class diagram

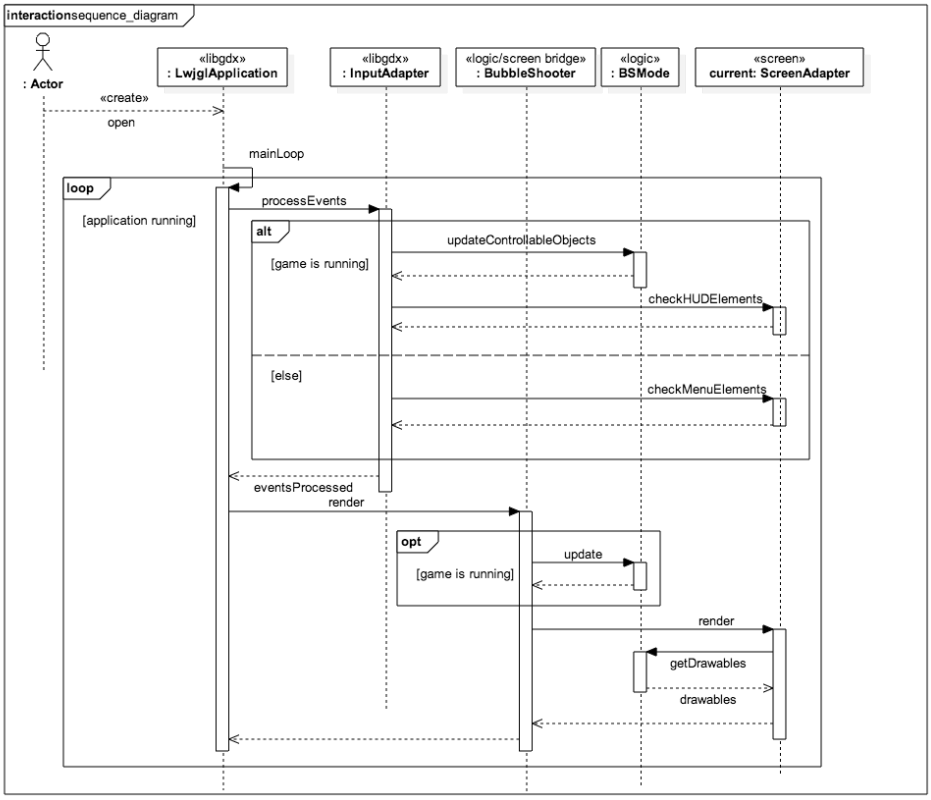

Figure 5: A snapshot moment from the new sequence diagram