Assignment 3

By group 30

1.1

For this iteration we choose the gaming experience as major point of improvement. The gems were appearing and disappearing way too fast and with no animation. In this sprint we've worked on that. The requirements are as followed:

Functional requirements:

Must have:

- The swap of gems in the game shall be an animation.
- The drop of gems after a found combination shall be an animation.
- The appearance of new gems shall be an animation

Should have:

- The removal of gems shall go smooth with an animation
- The game shall recognize when there are no options left and generate a game over.
- The Highscore layout shall have a background.
- The design of the pause menu shall be renewed.

Could have:

- The game shall have three special gems.
- The first special gem shall appear when a combination of 4 gems is formed. This special gem shall give a double amount of points when a combination with this gem is formed.
- The second special gem shall appear when a combination of 5 gems is formed. This special gem shall remove an entire row of gems when a combination with this gem is formed.
- The third special gem shall appear when a combination of 6 or more gems is formed. This special gem shall remove all gems in the field with the same color when a combination with this gem is formed.

Will not have:

- Animations with the build in animation function of javaFX

Non-Functional requirements:

- Animations will be done with a timer

1.2

For implementing the animation improvements we made use of UML and responsibility driven design. First we had a initial idea of how this should work. From this idea we've build further and made two UML diagrams, a class and a sequence diagram, to structure the idea before implementing it.

Initial Idea

Gemclass:
We'll give a gem a
boolean moving;
int animationpositionx;
int animationpositiony;
gem will still have a col and row position (or we might move this since board also holds these values)

There will be an animation class that does something like:

```
While(there are gems that are still moving)
        forall gems
                if moving
                        if (animationpositionx < col*dimension + offsetx)
                                animationpositionx ++
                        else if (animationpositionx > col*dimension + offsetx)
                                animationpositionx --
                        else
                                if (animationpositiony < row*dimension+offsety)
                                        animationpositiony++;
                                else if (animationpositiony > row*dimension+offsety)
                                        animationpositiony--;
                                else
                                        moving = false;
        small sleep
```

The general code should do something like this:

```
When mouseclick happened:
        set animationpositions on current position
        swap gem by changing col and row
        run animation
        check for combination
        no combination?
                swap back
        combination?
                delete combination, remove gems
                set animationpositions of all gems above the remove gems
                change rows to the new positions
```

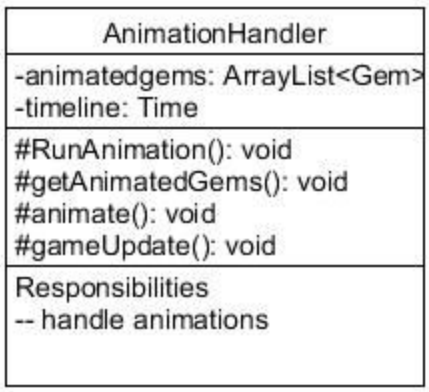
run animation

Check for combinations again and repeat till there are none.

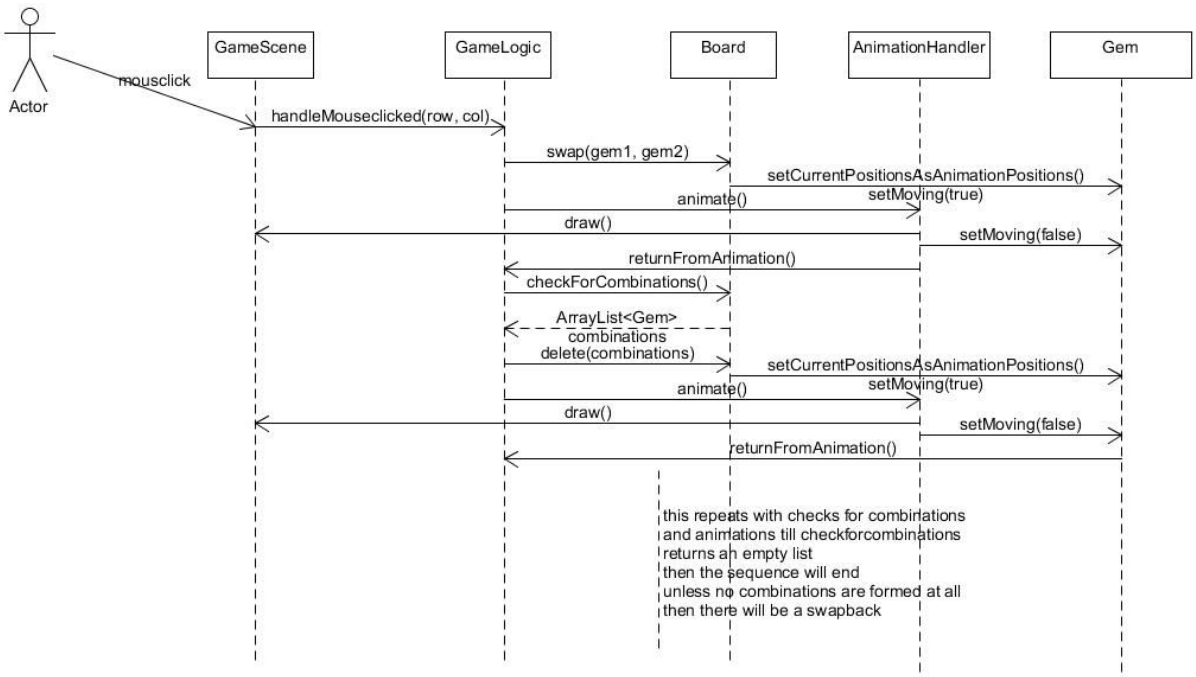
<u>During an animation:</u>

Disable mousehandeling.

ClassDiagram



Sequence Diagram



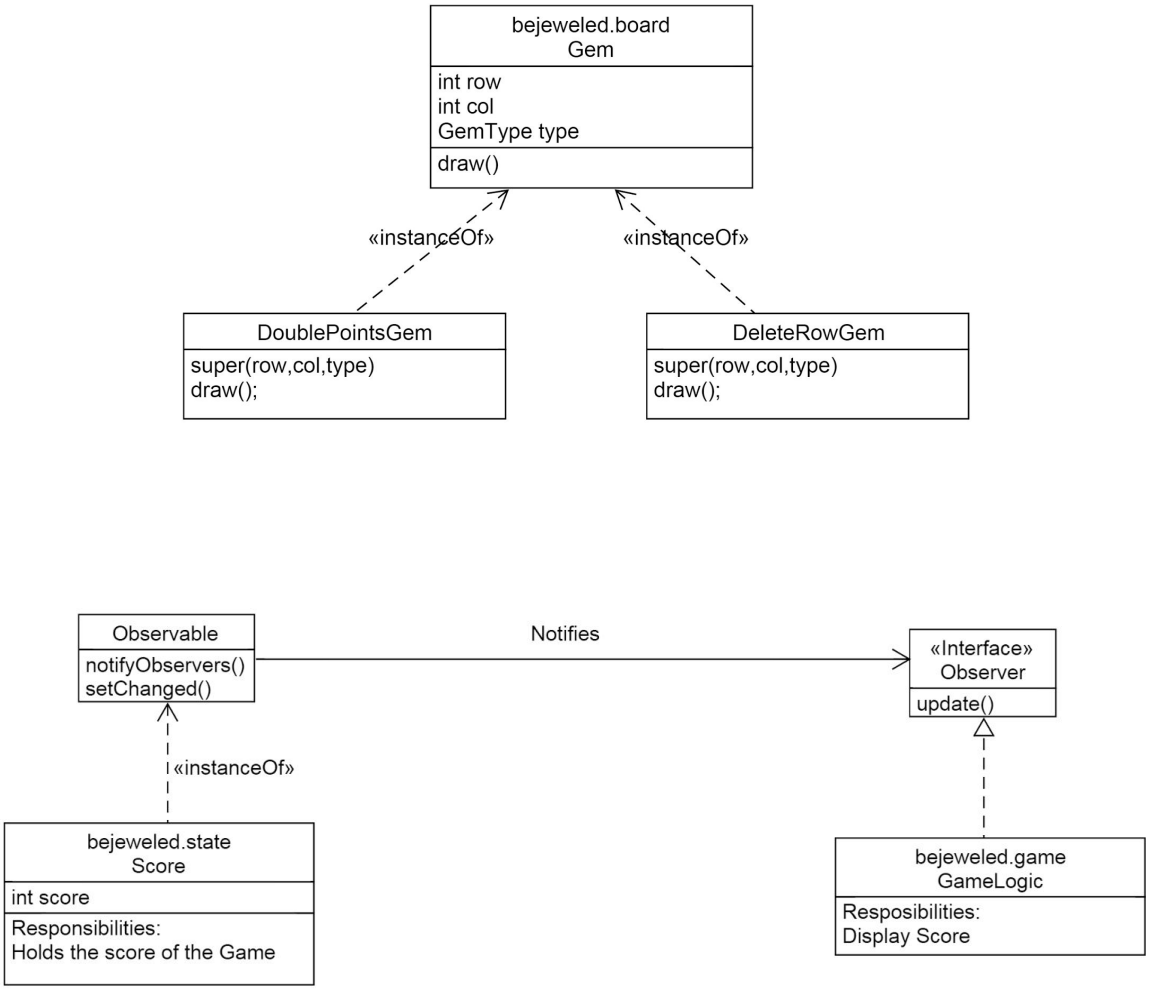
2.

Two design patterns are implemented in our Bejeweled game. The first one is the observer pattern and the second one is the factory pattern.

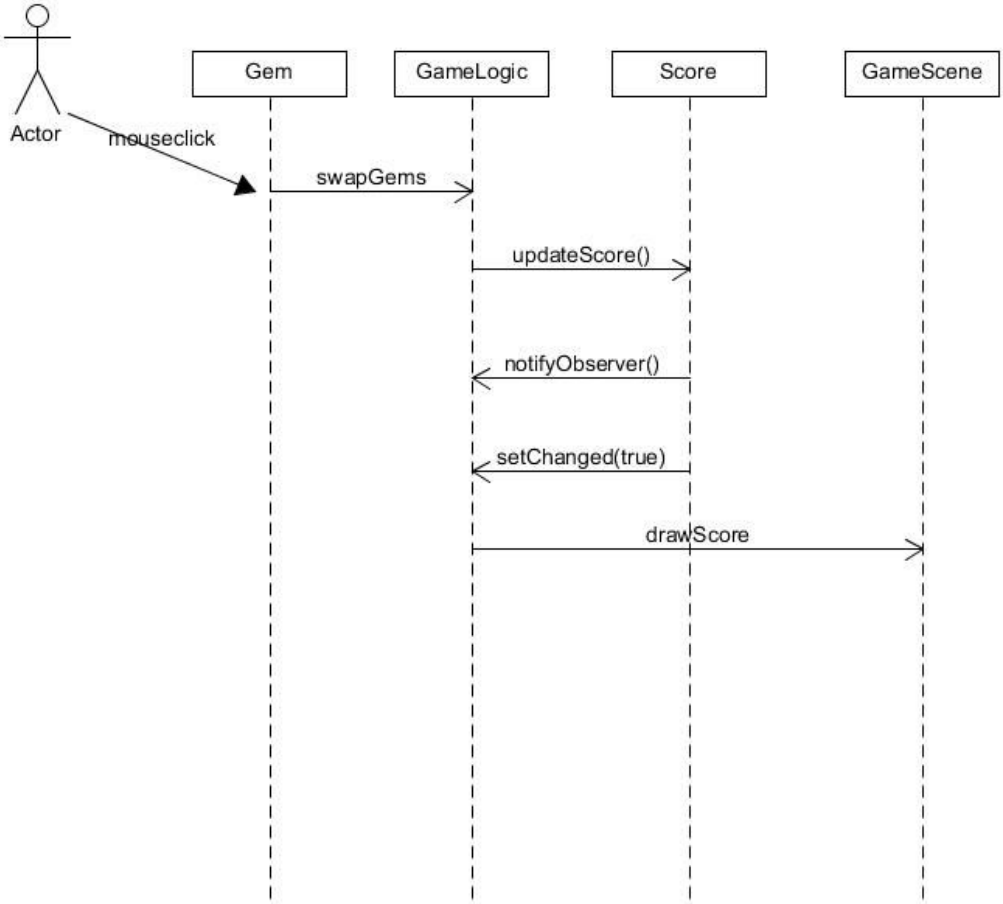
Observer pattern

The observer pattern is used on the Score. The class GameLogic displays the score in the GUI, so it is beneficial to make GameLogic an observer of Score. Now every time Score is updated (when the player combines multiple Gems), GameLogic is notified and will update the Score in the GUI. This is done by the methods setChanged() and notifyObservers() in Score and the method update() in GameLogic.This design pattern in made visible in the following UML diagrams.

Class diagrams



Sequence diagram

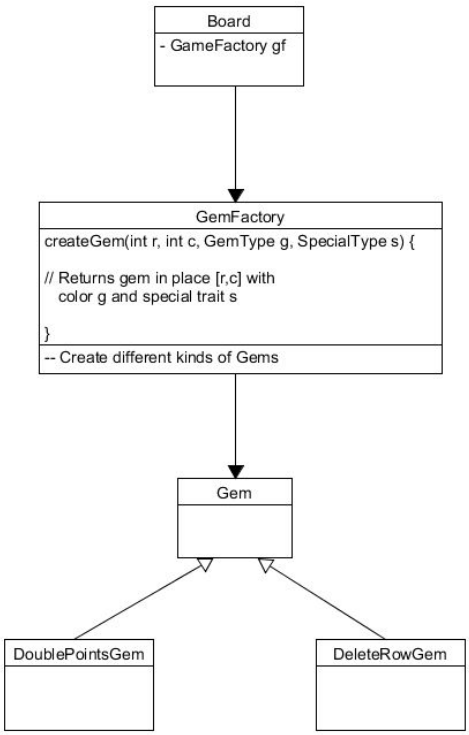


Factory design

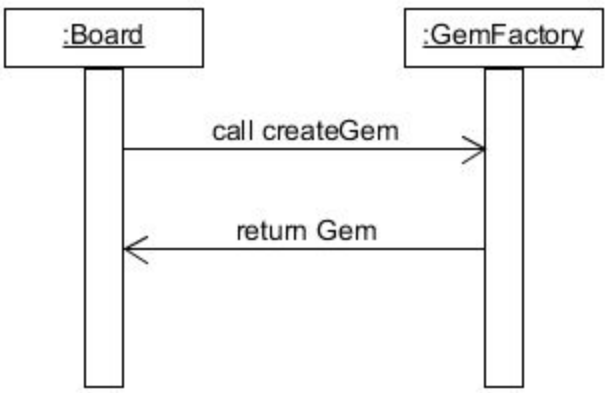
Our Bejeweled game will make use of ‘special gems’. We therefore have to create several different kinds of Gems. This way we can centralize gem creation by means of the factory pattern instead of intantiating all kinds of Gems by using “new <GemName>”.

We will make a GemFactory class which will take on the responsibilities of making Gems with the method createGem(int row, int col, GemType type, SpecialType special). Additionally we will make a new enum type: SpecialType. Depending on the SpecialType given to the createGem method, it will create a normal gem or one of the kinds of special Gem.

Class diagram



Sequence diagram



3.1

Explain how good and bad practice are recognized.

All projects are classified into four quadrants: Good Practice, Cost over Time, Time over Cost and Bad Practice. If for example a project factor occurs 50% or more in the Good Practice quadrant, the factor is strongly related to Good Practice. If a project factor occurs 50% or more in the Bad Practice quadrant it is a failure factor, the factor is strongly related to Good Practice.

3.2

Explain why Visual Basic being in the good practice group is a not so interesting finding of the study.

There were only 6 projects which used Visual Basic as the primary programming language and they took place in two different companies. Furthermore, the projects ranged from 27 to 586 FP. Five of the six projects scored as Good Practise and one a Cost over Time. Compared to the findings of experienced team (N=62), the dataset for Visual Basic is too small (N=6) to say that the stochastic findings are reliable.

3.3

Enumerate other 3 factors that could have been studied in the paper and why you think they would belong to good/bad practice.

- Group functions. This means having clear and strict functions in the group. For example someone who is in charge of all the code and or someone who is responsible for keeping up with the planning. The difference between a team with fixed and strict functions versus a team with changing functions seems to be an interesting study. We think having strict functions works better, so it's a Good Practice.

- Minimum cost development. Developing a program for the lowest costs. We think it is a Bad Practice, because testing will be minimized and the product will give a lot of bugs.

- Documentation. We think it is a Good Practice to have a lot of documentation during the process, so you can evaluate why certain decisions have been made.

3.4

Describe in detail 3 bad practice factors and why they belong to the bad practice group.

- Once-only project. This means doing a project of that sort only once, the group is inexperienced on the subject. The group does learn, but can not implement it in a next release. So this is a Bad Practice, because the team is very inexperienced in the matter and can make a lot of mistakes. And when the project has been delivered, the team will never make use of the things they've learned in this project.

- Rules & Regulations. Rules and Regulations limits the creativity and freedom of the team. Because the teams has to work with Rules and Regulations it can occur that it will take a long time to implement ideas according to the restriction. When it could have been done a lot faster. So the time aspect and the freedom limiting aspect together make it a Bad Practice.

- Many team changes, inexperienced team. If a team is inexperienced or changes during the process it will take a lot of time to get the whole team up to speed with the necessary knowledge. Time is money, so the duration of the project and costs will be higher. This together makes it a Bad Practice.