Sprint 3 Report

TI2206 - Software Engineering Methods

Karel van Doesburg
Robert Levenbach
Pascal Lubbers
Bas Stinenbosch
Sebastiaan Vermeulen

...presenting Tetris

Implementation of design patterns

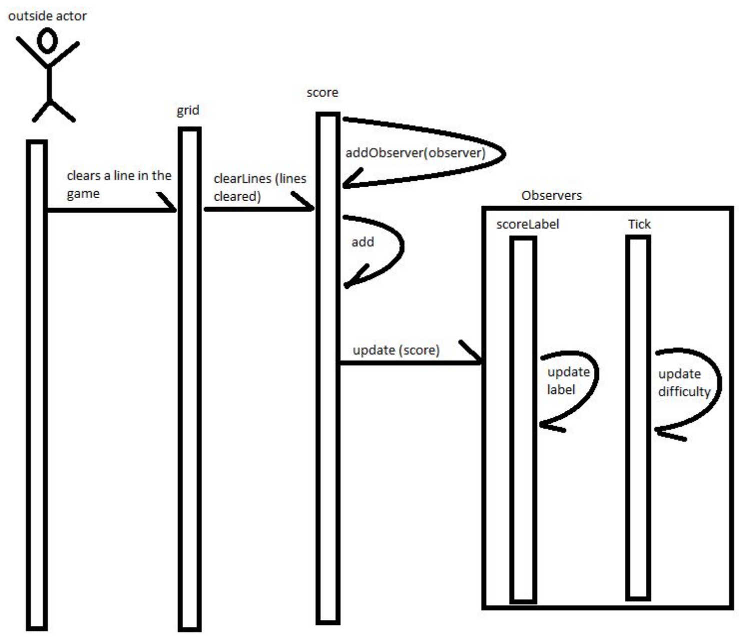
Subject-Observer design pattern

- An update of the score has implications for different elements of the game. The score counter has to be updated, the gravity has to be increased and during the development of the game more features may be added which must react to a change in the score. To properly track these changes we implemented the Subject-Observer pattern by creating a Score class (extending Observable) such that each object that should be updated by the Score can be easily added by extending Observer and implementing an update() method.
- Below it is shown which classes implement either the Observer interface or the Observable interface. When an integer is passed into the Add() method it will fire the Update() method of each registered observer. Therefore each of the classes that implement the Observer interface will respond the a score added in the Score class (the observable).

Tick implements **Observer**	**View** implements **Observer**	**Score** implements **Observable** **int** score

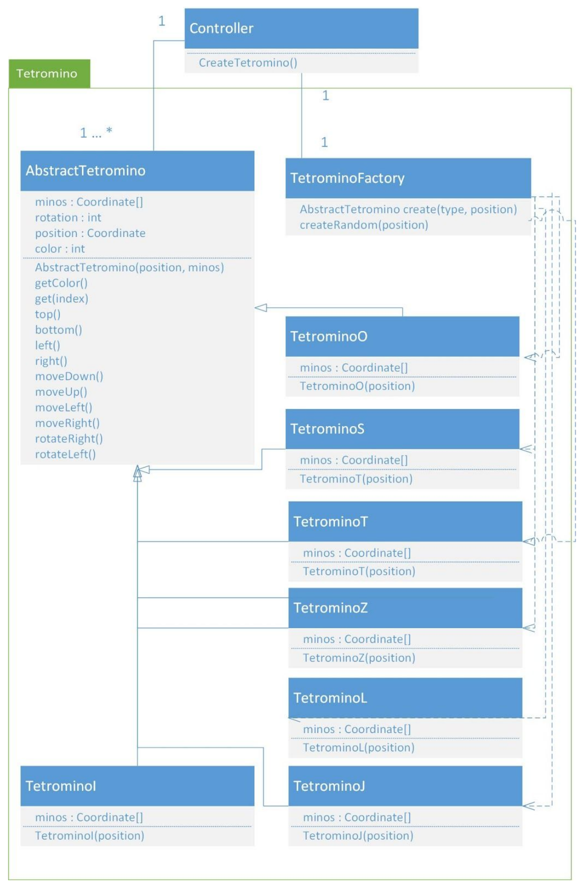
Update(Observable sender, Object arg)
...
	Update(Observable sender, Object arg)
	...
		Reset()
		Add(int add)
		Update(object arg)
		AddObserver(Object observer)

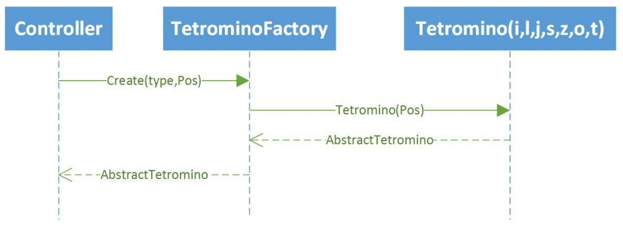
- In the following sequence diagram a scenario of interactions between the classes that implement either the Observer or Observable interface is visualized in temporal order. It is shown how this subject-observer construction responds to the input of the player (outside actor).



Factory pattern

- The creation of Tetrominos is an important part of the game. There are 7 unique Tetromino shapes that needs to be created in a random fashion. Therefore there needs to be some kind of abstraction that handles the creation of a random Tetromino. At this time the creation of a Tetromino is fairly simple but this could get more complicated over time. Implementing the creation of Tetrominos using the Factory pattern can help overcome these problems. The Tetromino ‘factory’ should have a method that accepts a Tetromino type en spits out an instantiation of the corresponding Tetromino. The factory can be extended - for example - with a method that creates a random Tetromino.
- The following diagrams visualize the implementation of the Factory pattern. It is shown that the Tetromino factory acts an abstraction layer between the Controller and the Tetromino shapes.





Preview Pane

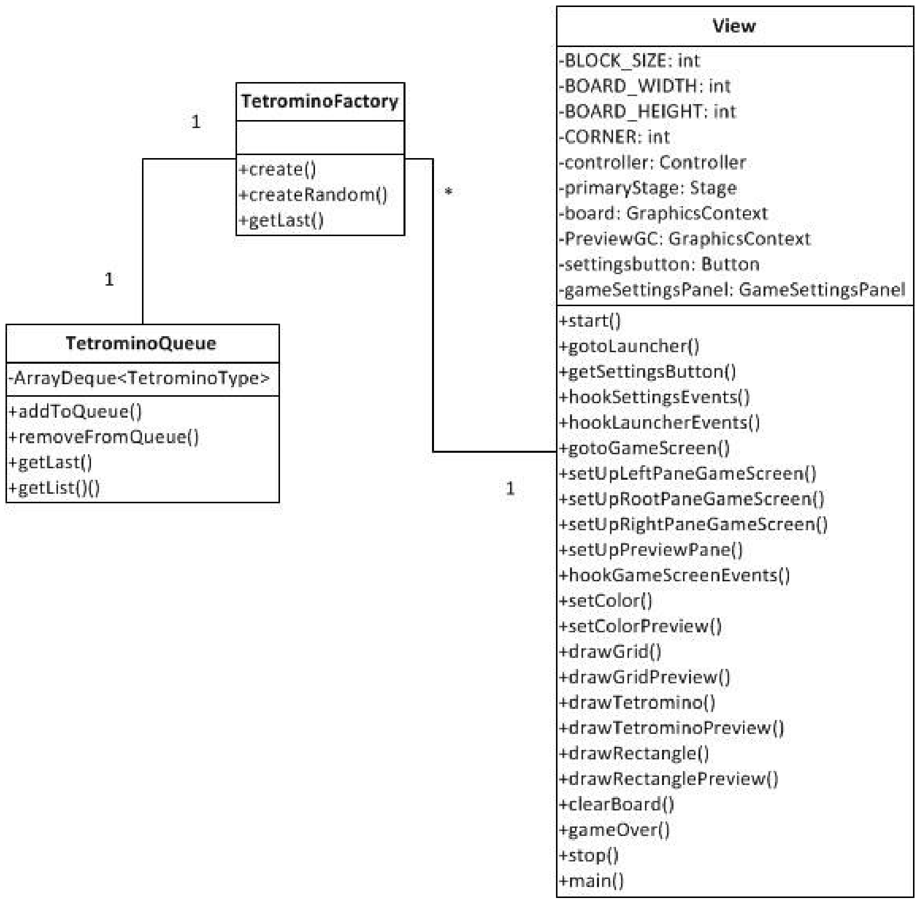
The preview pane is a small window next to the grid that shows the Tetromino that is going to be dropped directly after the active falling Tetromino has landed and is fixed in the grid. This could help the player to make a better decision where to place the active falling Tetromino and improve his/her way of stacking Tetrominos.

Requirement:

- The preview pane should show the next Tetromino to be dropped in the grid

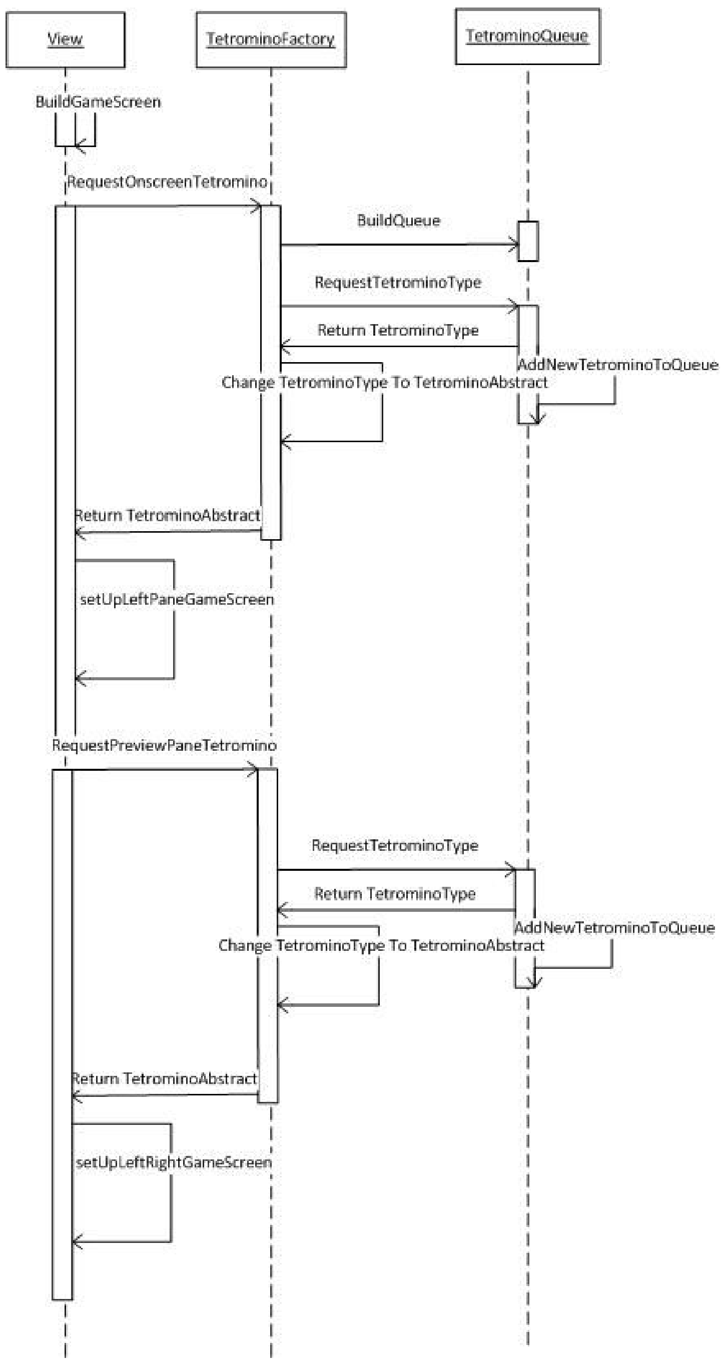
UML

As shown in the following diagram, the Tetromino queue holds a sequence of Tetromino instances. The factory acts as an abstraction interlayer that hides the queue implementation from the rest of the system.



Sequence diagram

This diagram show the interaction between objects that communicate with the Tetromino factory and the interaction between the Tetromino factory and the Tetromino queue as a sequence of action in temporal order.



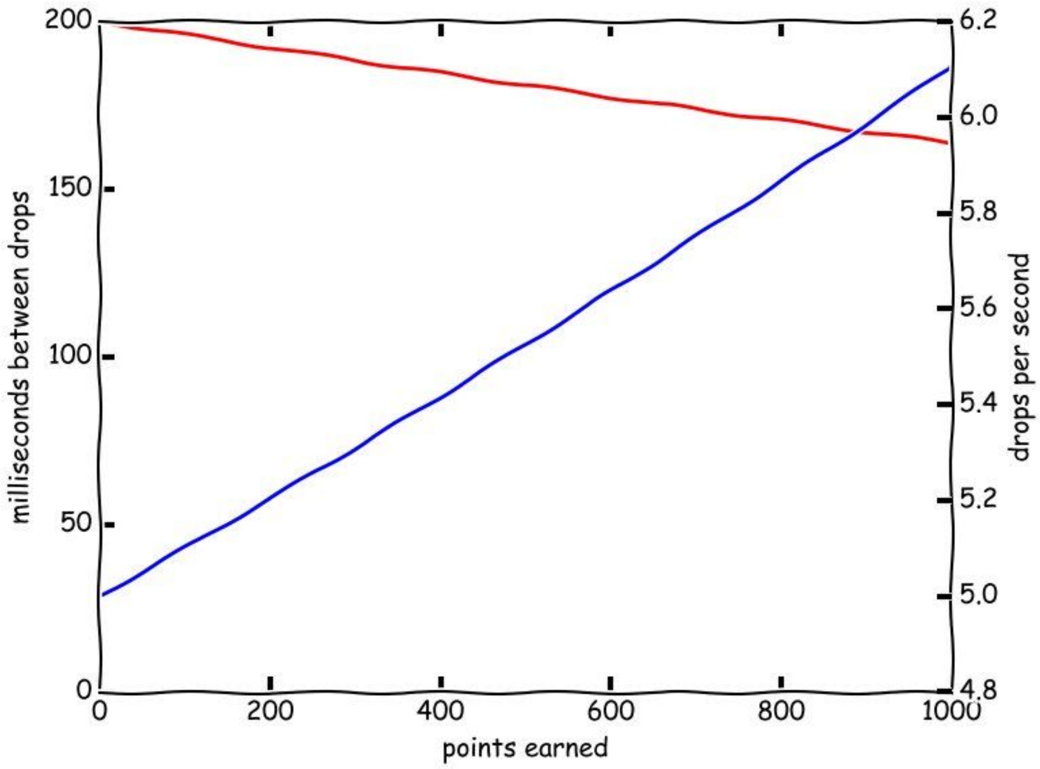
Scoring system

To make to game more competitive we decided to implement a scoring system. This system tracks the progression of the player for each individual game by counting the number of lines cleared. Each time the player clears a line, the score will increase. The following rules define the amount of points a player gets for clearing a number of lines at once. The more lines cleared at once to more points a player gets rewarded.

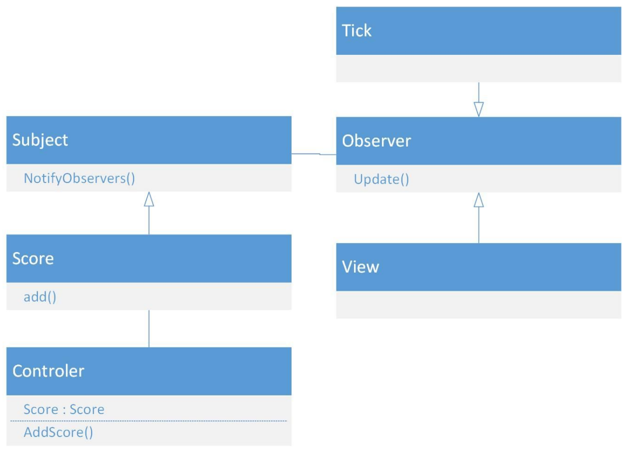
> 40 points will be given if 1 row of blocks is cleared
> 100 points will be given if 2 rows of blocks are cleared at the same time
> 300 points will be given if 3 rows of blocks are cleared at the same time
> 1200 point will be given if 4 rows of blocks are cleared at the same time

The number of lines cleared will affect the gravitational force that acts on the falling Tetromino. An increasing gravitational force means the falling Tetromino drops faster and therefor the will be less time to decide where to place the falling Tetromino. This will make the game more challenge as it progresses.

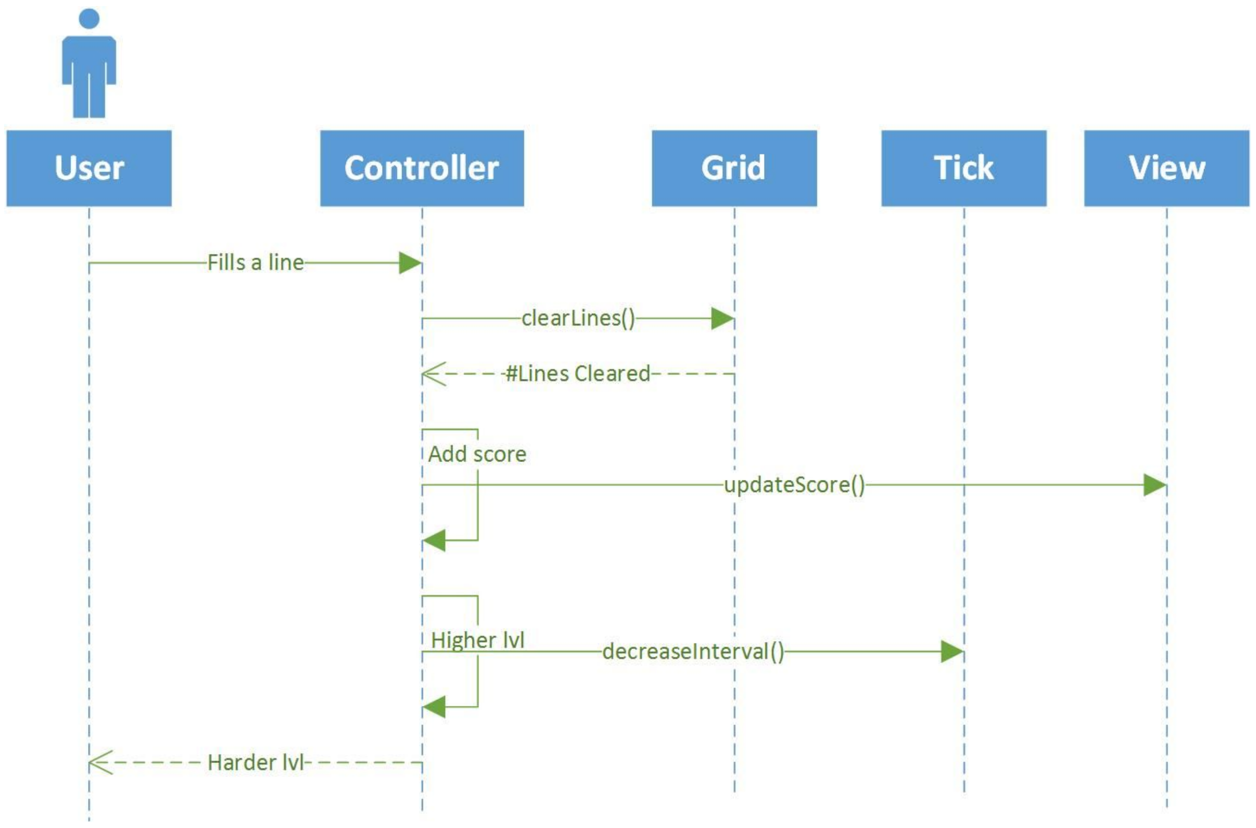
The following chart shows how the game speed increases each point earned.



When the add() method in the Score object gets called by the Controller each object that implements the Observer interface gets notified with the updated score. Each observing object will respond differently to this notification. The Tick object will decrease the time interval between ticks and the View will render the score for the player.



How the subject-observer system will respond to a score change is visualized in the following diagram.


User
Controller
Grid
Tick
View
Fills a line
clearLines()
#Lines Cleared
Add score
updateScore()
Higher lvl
decreaseInterval()
Harder lvl