WEEKLY ASSIGNMENT

ASSIGNMENT 1

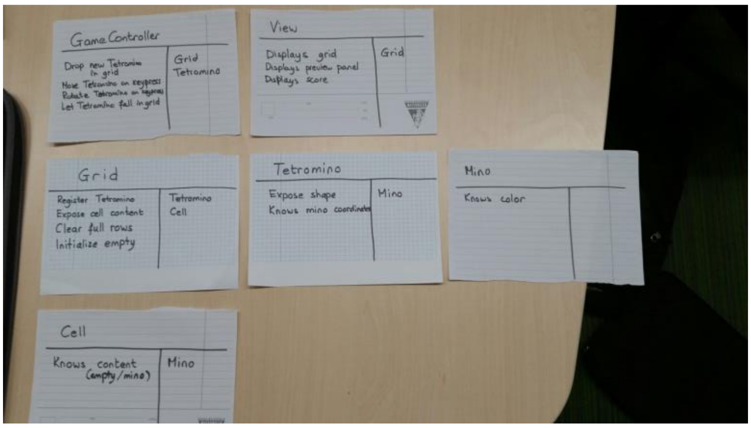
TI2206 Software Engineering Methods
Computer Science
Delft University of Technology

Robert Levenbach
Bas Stinenbosch
Karel van Doesburg
Pascal Lubbers

Exercise 1 - The core

CRC cards

To properly analyze our main classes and evaluate their functions we CRC cards to give us a clear overview, as can be seen below. The most important classes are at the top and the least at the bottom.



There are many differences between this design and the actual implementation. For example the Cell class, which is absent in the actual implementation. For know there isn't any added value over using an object for each cell in the grid versus just an array of integers indicating if a cell is empty or not.

Another difference is that we realized it would be a good idea to represent each Tetromino by an own class. For consistency and adhering to the DRY principle we constructed a base class containing all shared logic like rotation and movement of a Tetromino. Every unique Tetromino shape class is inherited from this base class.

Main Classes

GameController

Responsibilities

- **Drop new Tetromino in Grid:** The Gamecontroller class is the class that handles the tetromino's. It spawns a new tetromino when needed(beginning of the game and when a tetromino has landed). This happens with the *'drowNewTromino()'* function
- **Move Tetromino on keypress/Rotate Tetromino on keypress** : It also handles the playerinput. If you press an arrow to the left or right, it will move in the designated way. With the up and down arrow you can rotate the tetromino's. This uses *'checkRotateRight()/checkRotateLeft()'* functions.
- **Let Tetromino fall in grid:** This responsibility makes sure the tetromino falls in the grid. It uses the *'lowerTetromini()'* function. First it checks if it can lower(using the *' tryToLower()'* function, also located in de GameController class). Then it checks if you aren't gameover, checking if you haven't reached the top. This happens until it has reached the bottom.

Collaborations: The GameControllers class collaborates with the Grid class and the Tetromino class.

- **Grid**: It uses the Grid class to determine whether the grid is free(to lower as example). It also uses the height and width of the grid(which is specified in the Grid class), to determine whether a tetromino can move left/right/down.
- **Tetromino:** It uses all the Tetromino classes(every shape) to spawn a shape in the grid. Also with the player input left/right, it uses coördinates found in that class to move the tetromino

View

Responsibilities:

- **Displays Grid***:* The view class loads the specifications of the Grid class and uses these to draw the grid within the canvas using the *'drawGrid()'* function.
- **Displays preview panel**: Next to the Grid the view shall load the specifications of the new tetromino that shall be dropped in the game. We are still working on this function. The idea is that the preview panel will queue the next tetromino and the controller will spawn the tetromino from the queue and will then drop it.
- **Displays score:** The view class will load the score and display it underneath the preview panel. We Are also still working on this function.

Collaborations:

- **Grid:** The view loads the specifications of the Grid class and uses these to draw the grid within the frame.

Grid

Responsibilities:

- **Register Tetromino:** the grid needs to show the tetromino. In the function '*registerTetromino()'* the grid seeks if there is a tetromino, and sets the coördinate where there is a tetromino to the color of the tetromino.
- **Clear Full Rows:** if a row is full with tetrominos, the tetrominos to be deleted. This happens in the '*clearLines()*' function.
- **Initialize Empty:** in the *'clearBoard()'* function the empty grid is created.

Collaboration:

- **Tetromino:** the grid needs to no the coördinates of the tetromino.
- **Cell:** This class has now been integrated into the tetromino class.

Tetromino

Responsibilities:

- **Expose shape:** in the functions *'bottom(), top(), left(), right()'* the most bottom/top/left/right mino is put. If you have these mino's put in a coördinate, the shape is exposed.
- **Know mino coordinates:** in the *'Coordinate get()'* function, the coördinates of the tetromino are returned and updated, according to the new coördinates.

Collaborations:

- **Mino:** the m]Mino class gives the Tetromino class the color of the single Tetromino

Less Important Classes

Mino

The mino class has now been integrated in the Tetromino class.

Cell

The cell class has now been integrated in the mino class and therefore in the Tetromino class.

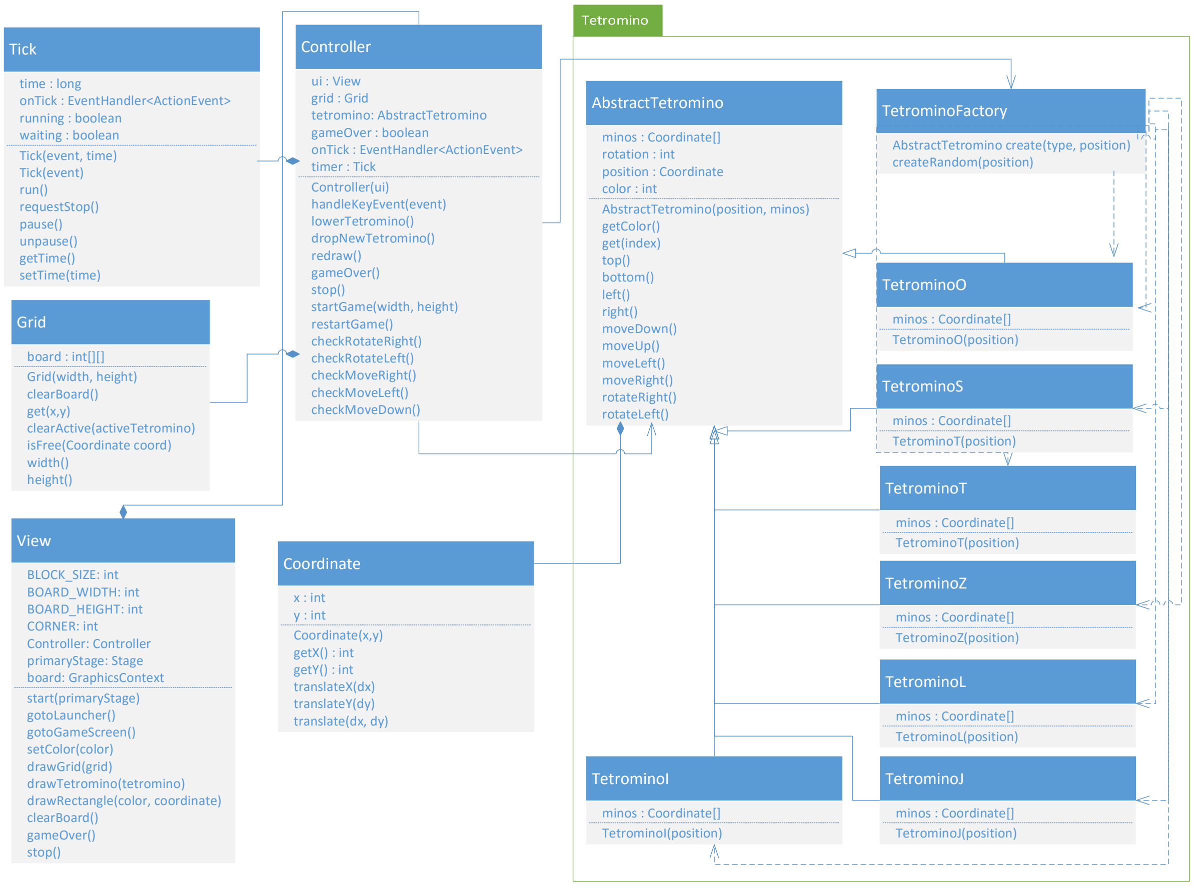
Exercise 2 – UML in practice

1. Aggregation is used where a class uses a class but can exist without it. A Composition is a class that uses a class. But if the class that's used doesn't exist the main class also can't exist.

   A Car has a driver (aggregation) and an engine (composition).

   An example of the game is that a controller has a grid, but without it it's not the game controller.

2. A parameterized class is a class that take's other classes as parameters. An example in the game is the abstract tetromino that uses a coordinate class.
3. Class diagram:



These are all the classes that are used in the game. In the tetromino package you can see the use of a factory design pattern. There are no hierarchies that should be removed.

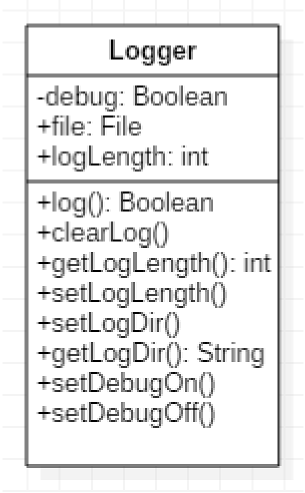
Exercise 3 – Simple logging

Functional requirements

- The logger should be able to log the object calling it.
- The logger should be able to log the summary of the action.
- The logger should be able to log the severity of the action.
- The logger should be able to write a log about an action to a file.
- The logger should be able to build up a buffer of logs to reduce I/O activity.

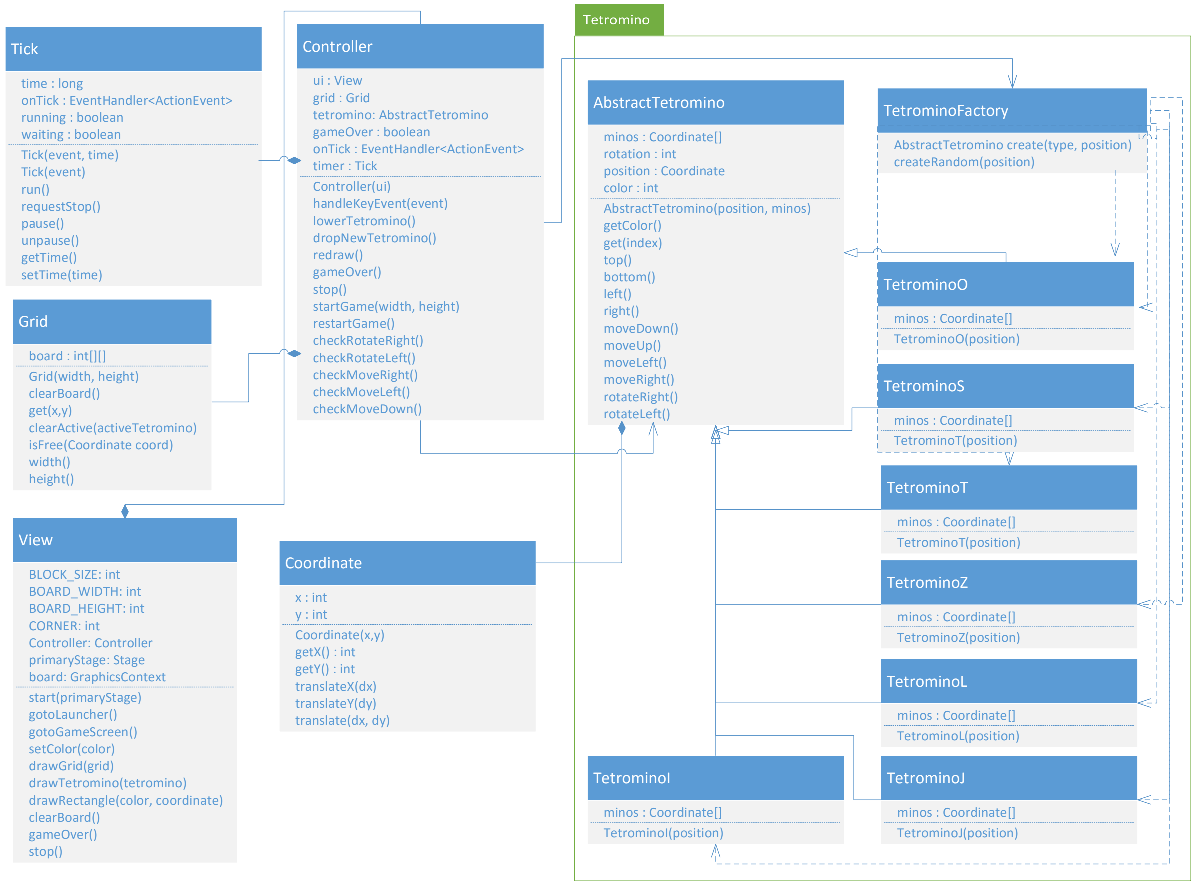
Responsibility Driven Design

For this feature we only need one class that is responsible for all logging related business. Other objects collaborate with this Logger class by calling its logging functionality.

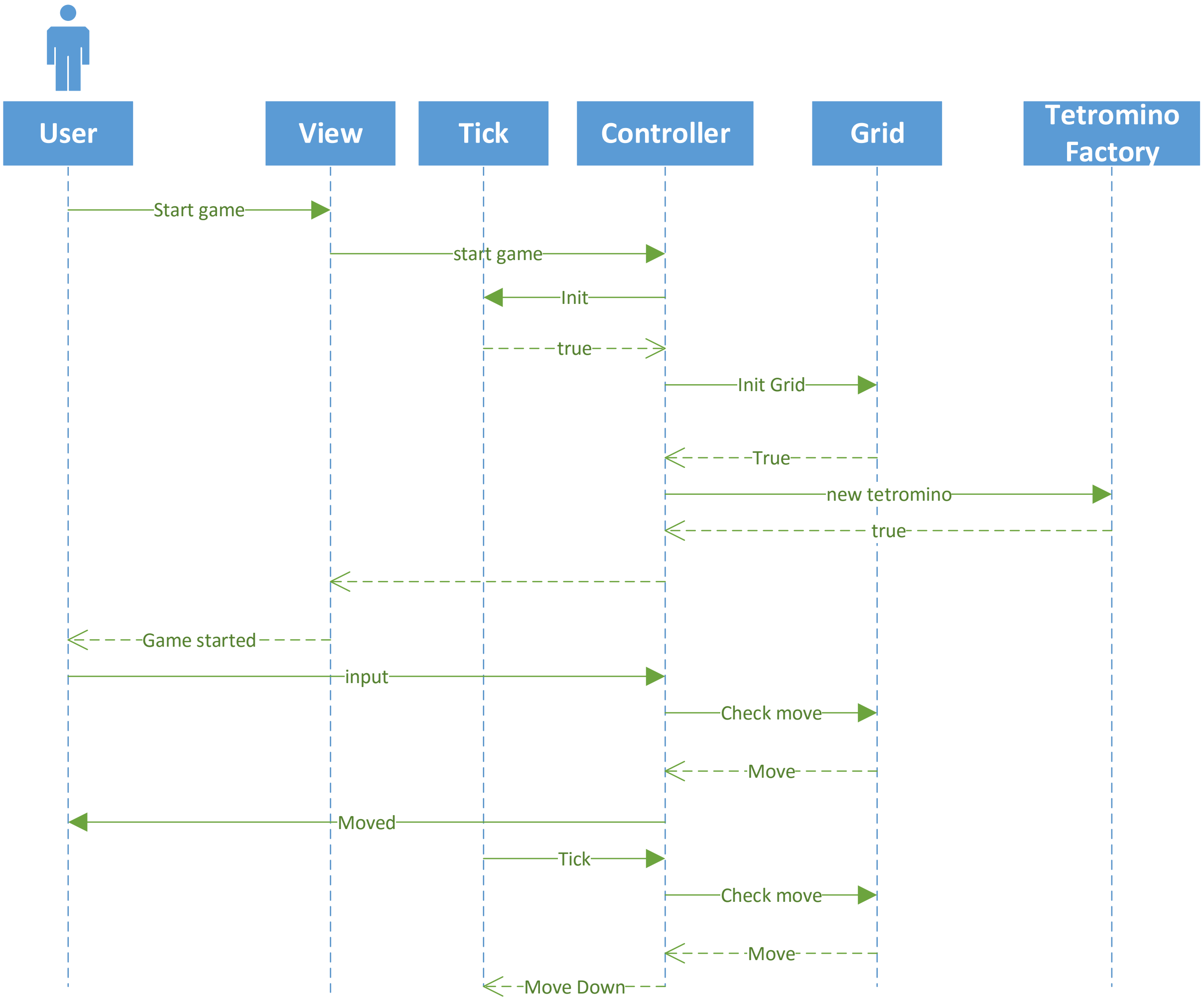


Appendix A

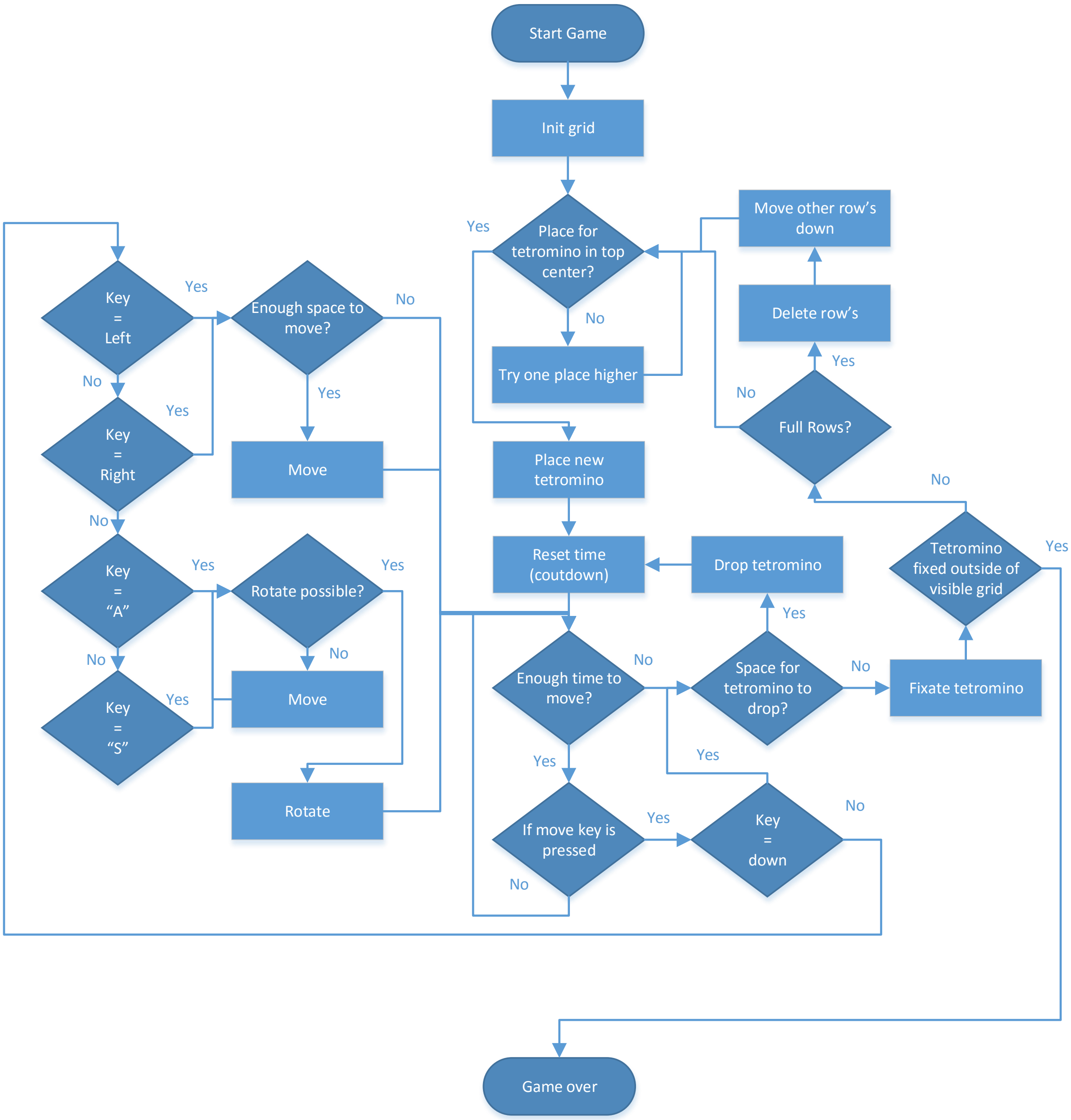
UML



Activity diagram



Flow Chart


Start Game
Init grid
Place for tetromino in top center?
Yes
No
Try one place higher
Place new tetromino
Reset time (coutdown)
Move other row’s down
Delete row’s
Yes
No
Full Rows?
No
Tetromino fixed outside of visible grid
Yes
Drop tetromino
Yes
Enough time to move?
No
Space for tetromino to drop?
No
Fixate tetromino
Yes
Yes
If move key is pressed
Yes
Key = down
No
No
Key = Left
Yes
Enough space to move?
No
No
Key = Right
Yes
Yes
Move
No
Key = “A”
Yes
Rotate possible?
Yes
No
No
Key = “S”
Yes
Move
Rotate
Game over