Explanation for Cube.ts, Fragment.ts and Main.ts

Script Cube.ts:

The script *Cube.ts* generates the smallest part of a fragment - a cube.

Since each cube is to be assigned a colour, the names of the colours are stored in the enum `CUBE_TYPE`. Thus, the names are also saved against write errors that can occur in a string.

In addition, the variable `Materials` of the generic data type *Map* is created to assign a material to each colour.

The class `Cube` inherits from `f.Node`, since every cube should also be a node. With `super` the node gets a name.

The class knows two variables - `mesh` and `materials` and the functions `createMaterials` and `constructor`. `createMaterials` fills the variable `Materials` with the appropriate key-value pairs.

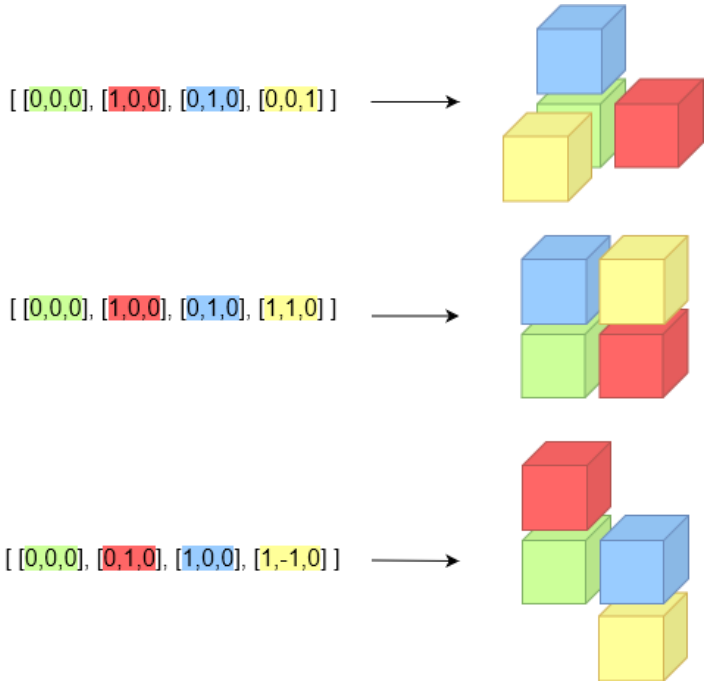
The `constructor` function takes a colour and a position and generates a cube. In order for the cube to be assigned the appropriate colour, `Matierials` searches the corresponding material using the colour as key. At the end, the cube is shrunk a little bit to create a demarcation to a possible neighbouring cube.

Script Fragment.ts:

Fragment.ts generates a fragment from the cubes generated with *Cube.ts*.

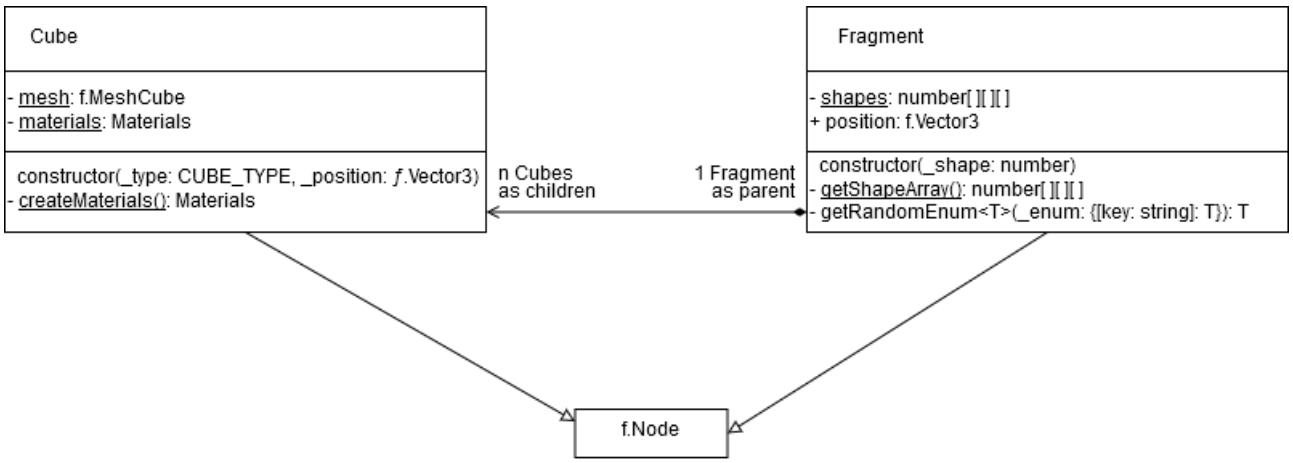
The class `fragment` also inherits from `f.Node` and gets a name via the `super` command. The class knows the variables `shapes` and `positions`, as well as the functions `constructor`, `getShapeArray`, and `getRandomEnum`. `shapes` is a three-dimensional array filled by the `getShapeArray` function. After that, `shapes` knows all sorts of shapes that a fragment can take on. The shapes are available as two dimensional arrays in which the positions of the individual cubes are stored.

The graphic shows how the fragment looks to the respective array:



The `constructor` function takes a number that selects a shape for the fragment. Then the fragment is built out of the cubes with a for-of-loop. To do this, the `getRandomEnum` function selects a random value from `CUBE_TYPE`, saves the position of the cube as a vector, then generates a cube with the appropriate colour and position by calling `let cube: cube = new cube (type, vctPosition)` and adds the cube to the fragment.

The class diagram illustrates the relationship in which *Fragment.ts* and *Cube.ts* are related to each other:



Script Main.ts:

As usual, a viewport and a main node are created in *Main.ts*, as well as a canvas and a camera in the `hndLoad` function.

Passing the boolean variable `true` to `f.RenderManager.initialize` causes antialiasing. Thereafter, three fragments are generated, moved to different positions and attached to the main node.

In addition, *Main.ts* has the function `hndKeyDown`, which registers a key press and rotates all three fragments in the corresponding direction. For the rotation, first the value of the variable `rotate` is changed and then the changed value of the rotation of the fragments is assigned. At the very end the image is refreshed with `f.RenderManager.update` and `viewport.draw`. Thus, a new picture is not constantly drawn.