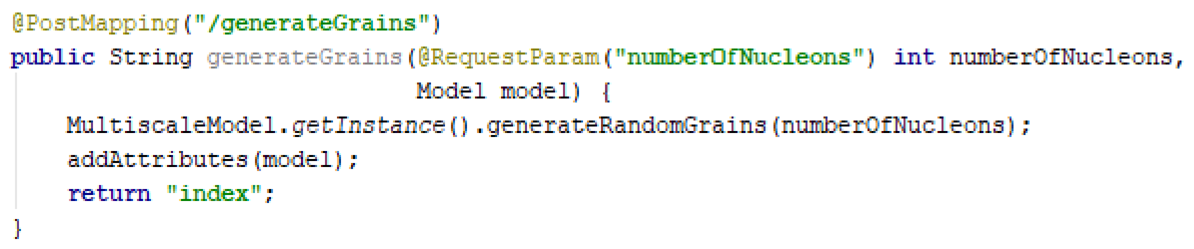
	Oliwia Strzelec	Date of execution 10.10.2018
Mutliscale Modeling	1st report	Date of commissioning 13.11.2018

1. Technologies

1.1. Spring

The simulating grain growth application was written in web technology called Spring. Spring is an open source application framework for the Java platform. It let me set up the web application. The Spring Web framework provides Model-View-Controller architecture and ready components. The MVC pattern results in separating the different aspects of the application. The Model part is whole logic side of the application happens in the backend and is written in Java language. The View generates HTML output. The Controller processes user requests and builds appropriate model, then passes it to the view for rendering. The Controller example is shown below.



```
@PostMapping("/generateGrains")
public String generateGrains(@RequestParam("numberOfNucleons") int numberOfNucleons,
                             Model model) {
    MultiscaleModel.getInstance().generateRandomGrains(numberOfNucleons);
    addAttributes(model);
    return "index";
}
```

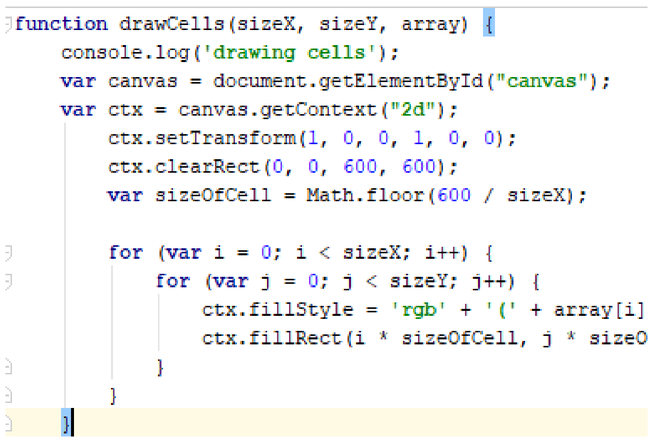
Picture 1.1.1. The example of controller handling the View form

1.2. Thymyleaf

Thymeleaf is a server-side Java template engine. Thanks to Thymeleaf HTML backend side can be correctly displayed in many browsers. The attributes from backend are easily passed to the frontend side. Using the Model class entire cell array is send to frontend and shown using JavaScript.

1.3. JavaScript

JavaScript is a high-level interpreted programming language, essential for using the World Wide Web. It was used only for graphic representation of the cell array passed through Thymeleaf attributes. Every time the site is loaded function shown below draws the table.



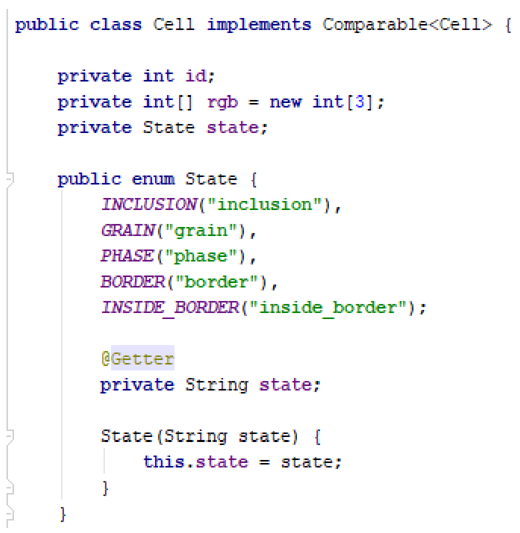
```
function drawCells(sizeX, sizeY, array) {
    console.log('drawing cells');
    var canvas = document.getElementById("canvas");
    var ctx = canvas.getContext("2d");
        ctx.setTransform(1, 0, 0, 1, 0, 0);
        ctx.clearRect(0, 0, 600, 600);
        var sizeOfCell = Math.floor(600 / sizeX);

        for (var i = 0; i < sizeX; i++) {
            for (var j = 0; j < sizeY; j++) {
                ctx.fillStyle = 'rgb' + '(' + array[i]
                ctx.fillRect(i * sizeOfCell, j * sizeO
            }
        }
    }
```

Picture 1.1.3. Function which draws cell table in view

2. Issuses

The purpose of the application is to simulate the formation of steel microstructure using cellular automata. This means that the basic operational unit of the application is a single cell. The cell has a greater than zero value for each nucleon Id and an RGB colour table. In addition, the cell may have a state different from the initial state (which is grain). The states are open to extensions depending on future design requirements.

The cells are stored in a two-dimensional array, which is at the beggining initialized as an array of cells with index 0. The first decision an application user can take is the number of nucleons to be generated and randomly placed in the array. Cells that become nucleons gain a unique Id and colour, which allows them to grow.

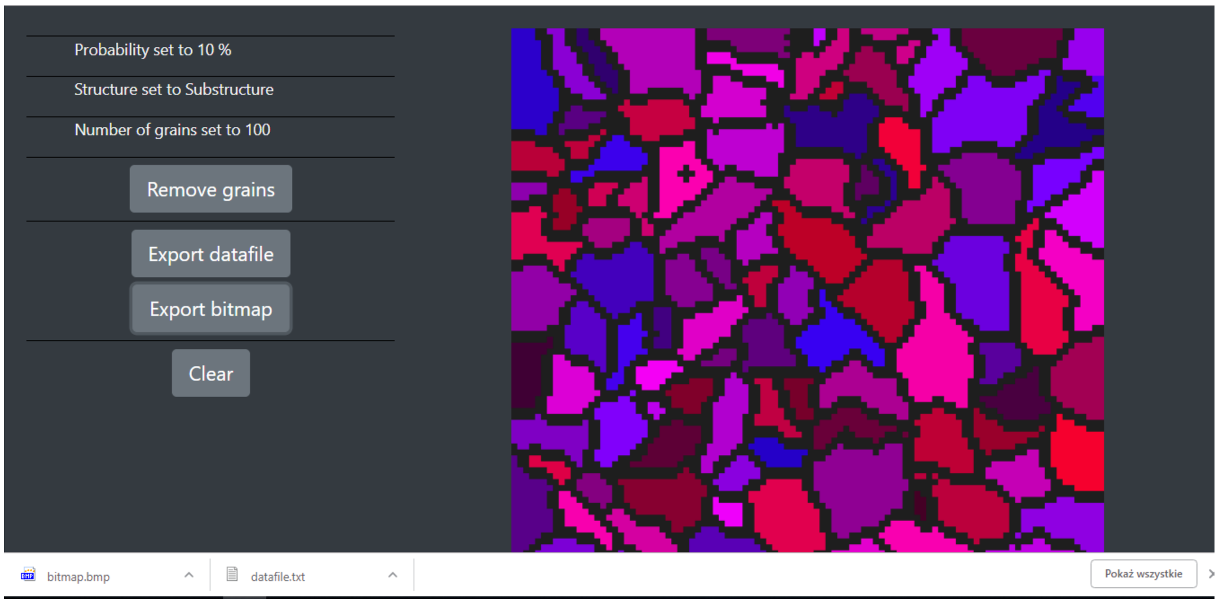


```java
public class Cell implements Comparable<Cell> {

    private int id;
    private int[] rgb = new int[3];
    private State state;

    public enum State {
        INCLUSION("inclusion"),
        GRAIN("grain"),
        PHASE("phase"),
        BORDER("border"),
        INSIDE_BORDER("inside_border");

        @Getter
        private String state;

        State(String state) {
            this.state = state;
        }
    }
```

Picture 2. Cell class which is the base of the project

2.1. Microstructure export

The basic functionality that the application is equipped with is the ability to export data from the cell table to a .txt or .bmp file. Due to the fact that the application is web-based, the files after export are stored in user’s downloads folder (according to his settings). The .txt file contains information about the x-coordinate, the y-coordinate, the id, the r, g and b colour constants and the state of the cell. The .bmp file has the actual size of the array of cells in pixels.



Probability set to 10 %

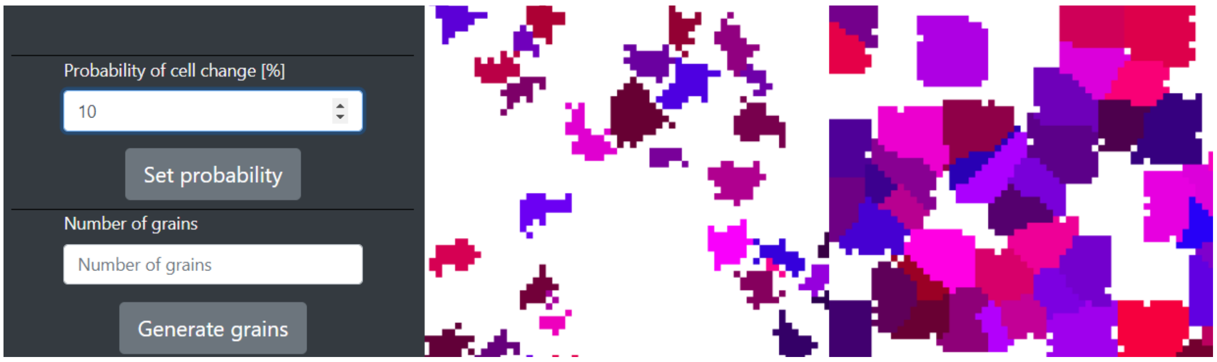
Structure set to Substructure

Number of grains set to 100

Picture 2.2.1. Graphic interface with buttons for export and exported files "bitmap.bmp" and "datafile.txt"

2.2. Grain boundary shape control

Cell growth happens using the neighbourhood of Moor and the four principles described below. Initially, when there is a small number of nucleons and a large number of free fields, the fourth principle is the most important. It leads to growth even in the presence of only one neighbour, with a predetermined probability. Probability is a parameter that can be chosen by the user. Otherwise, it adopts a default value of 90, which provides results closer to the original implementation of the cell proliferation algorithm using neighbourhood, and thus regular growth.


Probability of cell change [%]: 10

Number of grains: Number of grains

Picture 2.2.2. From the left: user's form for choosing the probability, cell increment with probability equal to 10, cell growth with probability equal to 90

a. First rule

The first one is important at the moment when a large part of the board is completed. It allows you to change the state of the cell only when at least 5 of its neighbours (from the neighbourhood of Moor) are representatives of the same grain. Otherwise, the second rule is executed.

b. Second rule

The second principle is analogous to the first, with the difference that it checks von Neuman's neighbourhood. If at least three neighbours are representatives of the same grain, there is a growth.

c. Third rule

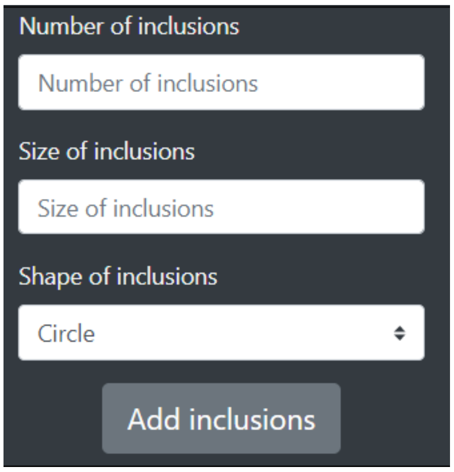
The third rule examines the neighbourhood as a result of subtracting von Neuman's neighbourhood from the neighbourhood of Moor. It checks whether the number of neighbours in a given type is greater than or equal to three.

d. Fourth rule

The likelihood of the implementation of the fourth method is determined by the probability parameter. The fourth rule is the initial implementation of the grain growth algorithm. It checks to see if there is a cell with id more than zero (that is grain) for all neighbours in the Moor neighbourhood. If so, it stores information about these cells in the list, and then selects the most common representative. After selecting the winner, the value is assigned to the empty cell.

2.3. Inclusions

The purpose of the implementation was to add inclusions - round or square objects of a given size, which are permanently placed in the table. In the case of inclusions in the shape of a circle, the given size is its radius, while in the case of a square - the length of the diagonal. The cell in the state of inclusion is dark coloured and has changed id to negative, to protect it from taking part in the growth.



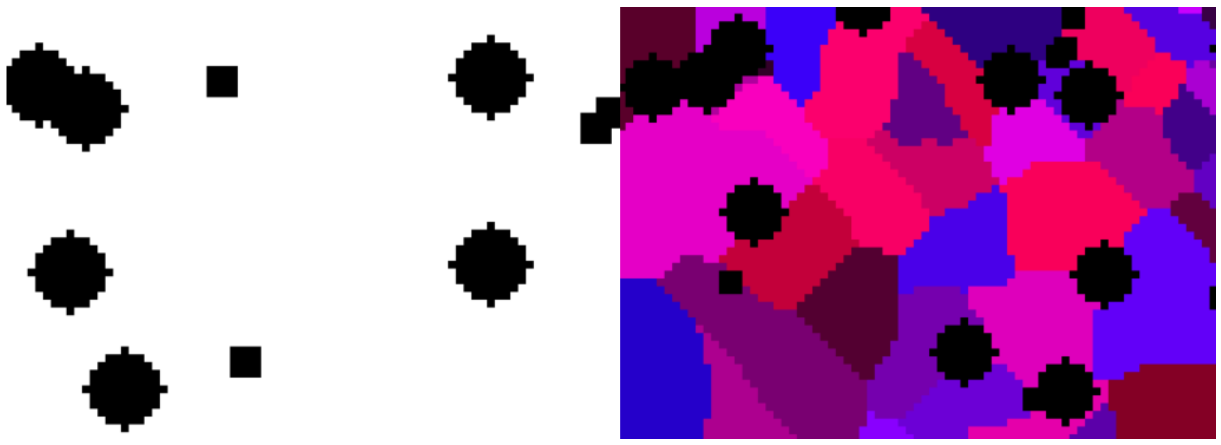
Picture 2.2.3. Form which allows to add inclusions

a. At the beginning of the simulation

When adding inclusions, two cases were distinguished - when adding takes place before the generation of nucleons, and after the extension. In the first case, inclusions are randomly generated in the entire empty array of cells and can be generated many times. Inclusions can overlap.

b. At the end of the simulation

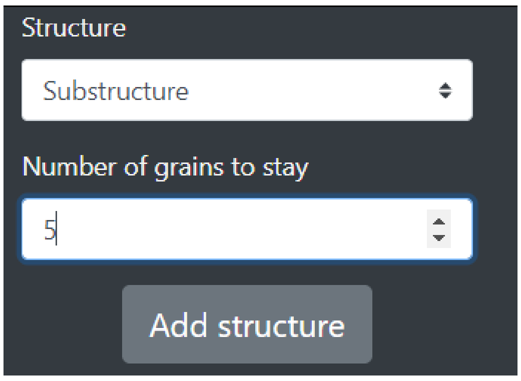
In the second case, after completing the whole table, inclusions can only be added at grain boundaries.



Picture 2.2.3.b. From the left, inclusions added at the beginning, inclusions added at the end

2.4. Substructures CA

Another feature was the addition of the substructure. After completing the whole table in the first growth, it is possible to draw grains that are to remain for the next growth. The grains drawn during the second growth retain their form and do not change in size.



Structure

Substructure

Number of grains to stay

5

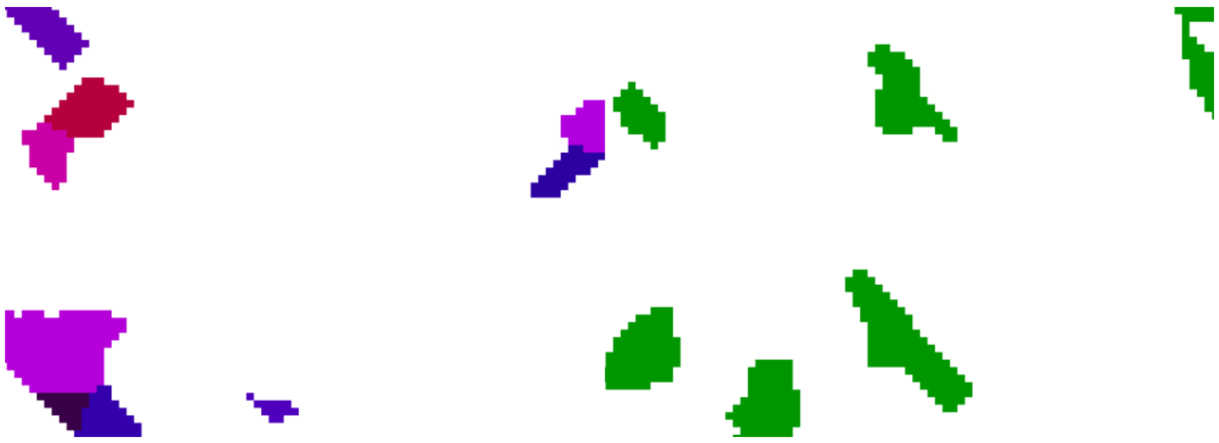
Picture 2.2.4. Form which allows to add substructures

a. Substructure

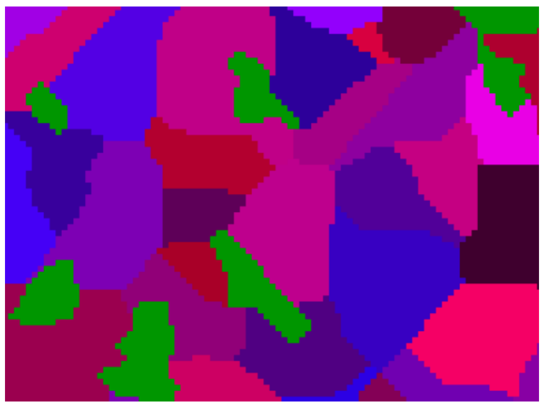
The first variant of the above functionality consists in the fact that the seeds do not change their colour. However, they change their id, which protects them from participating in the next growth.

b. Dual phase

In the second case, grains drawn as substructural elements change their phase, that is they receive a new record about the state and the id and colour, which makes it easy to distinguish them after the expansion.



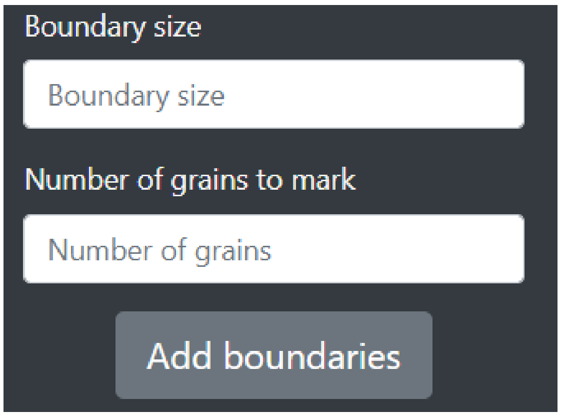
Picture 2.2.4.b$_1$. From the left: generated substructure, generated dual-phase substructure



Picture 2.2.4.b$_2$. Completed array with dual-phase substructure

2.5. Boundaries coloring

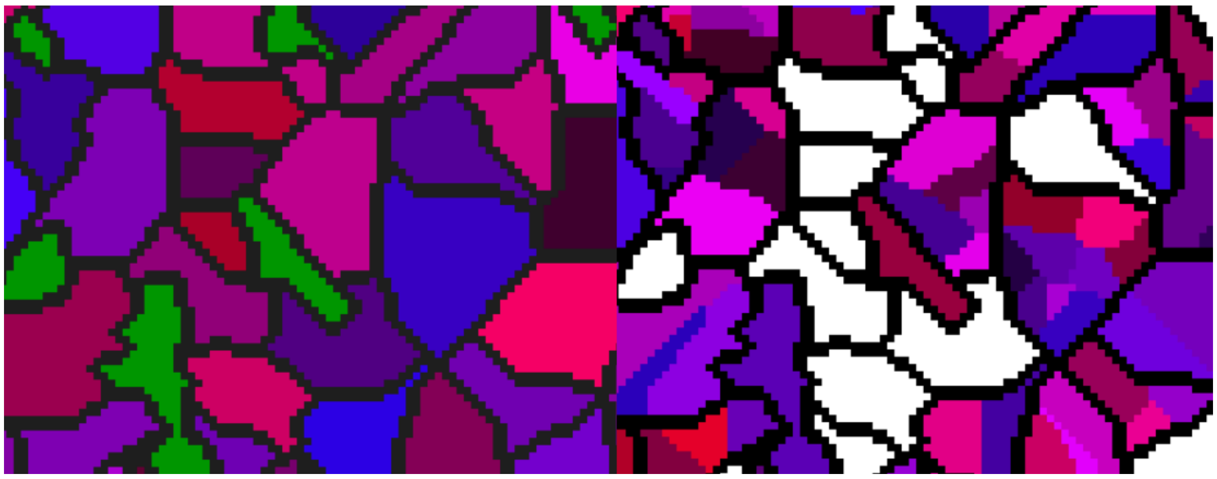
The last implemented feature is the ability to generate grain boundaries after completing the table. The form for generating borders has two fields - one responsible for the size of the boundaries and the other - for the number of grains to be surrounded by boundaries. Behaviour during the second growth depends on whether all the seeds have been selected in this step or not.



Picture 2.2.5. Form which allows to add boundaries

a. All grains selected

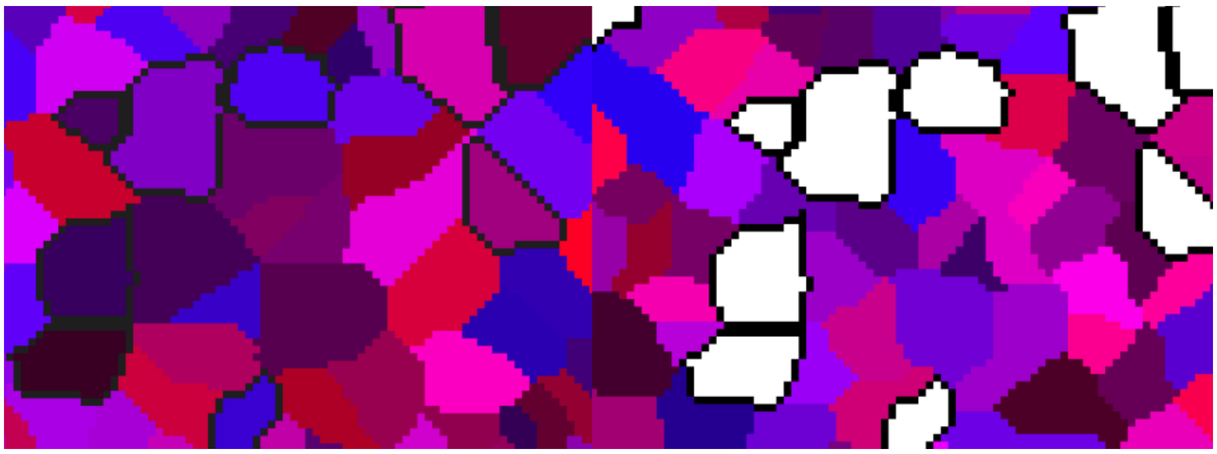
If the number of grains is not entered and the field is left blank or when the value 0 is entered, the boundaries of all grains are set up. In the next step, using the "Remove grains" button, you can get rid of grains from the inside of borders. At this point, the borders become inclusions and a second growth is possible. You can re-draw the amount of grains and start filling the board. It is possible that some elements of the board will remain empty due to the fact that it is not possible to proceed growth inside the borders in which no grain has been drawn.



2.2.5.a. From the left: all grains surrounded by borders, the second growth made after leaving only the borders of all grains with visible empty spaces in places there were no nucleons

b. N grains selected

In case a specific number of grains is selected, they are randomly marked. Then the cells at the borders are assigned the "border", and the "inside border" state is assigned to the cells inside the borders. Thanks to this, in the next step, you can clean the table and prepare it for the next growth, without losing information about the cells that are inside the borders. The generation of nucleons during the second growth is carried out with the exception of the space occupied by grains surrounded by borders.

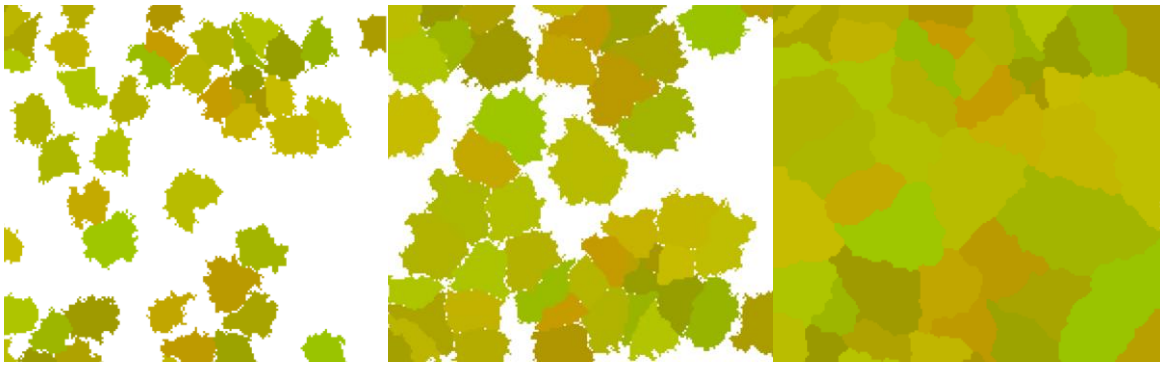


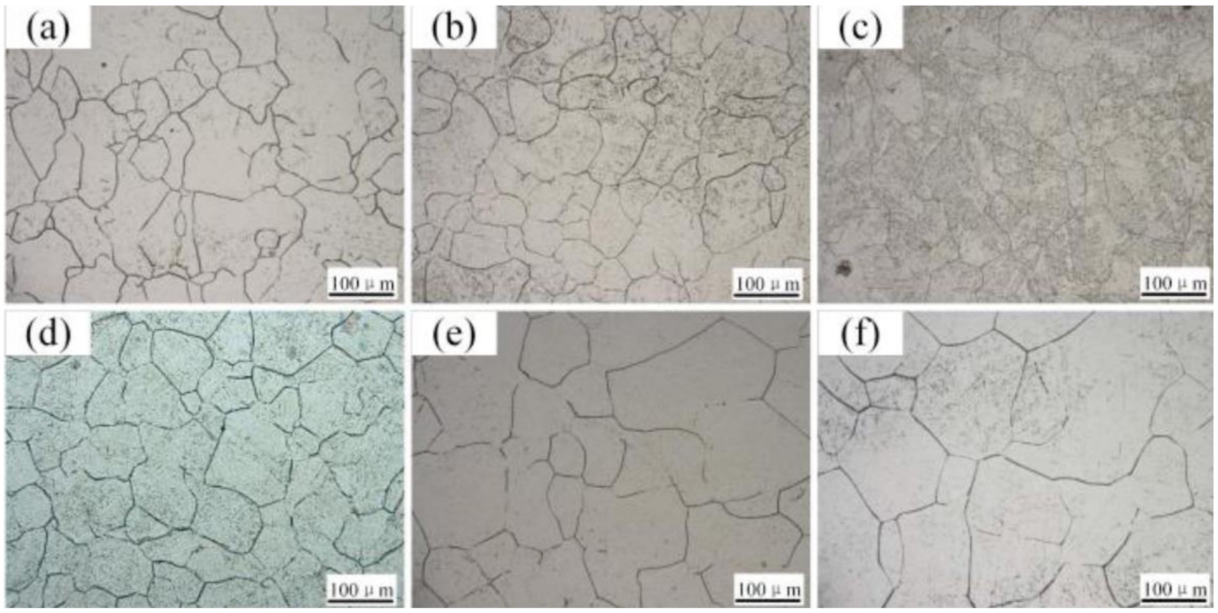
Picture 2.2.5.b. From the left: An array with several grains surrounded by borders, a table after the second growths

3. Grain growth in literature

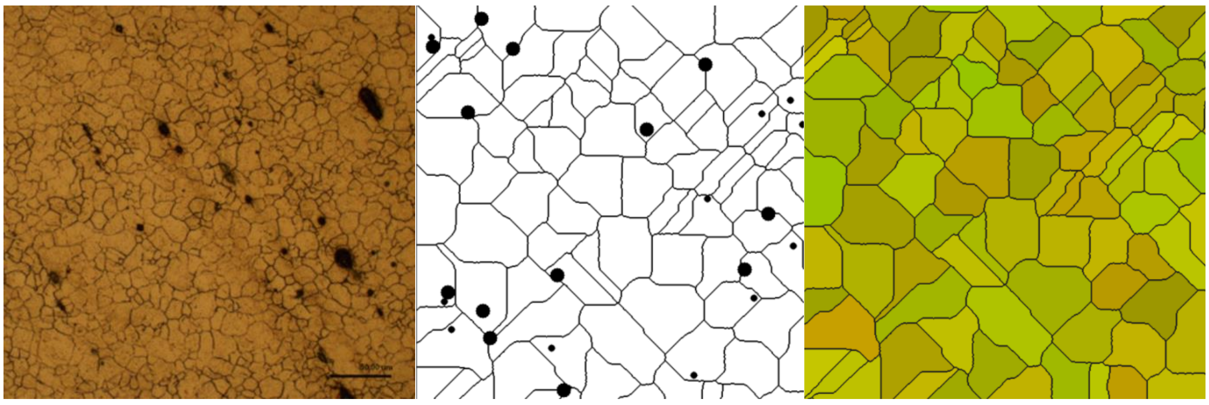
Most publications and studies deal with the effects of heating/cooling rates and the length of holding the alloy at a given temperature. The influence of these variables on the shape and size of grains as well as the shape and size of intergranular borders are investigated.

In the realistic picture of the structure, the boundaries are much thinner in relation to the size of grains. This can be changed if tables with more cells are created. On realistic photographs, the inclusions are barely visible, in the application they are big and clearly visible. Our model does not take into account that borders may disappear in places. In the application, the color of the grain was drawn from the RGB palette, while the realistic pictures of the structure usually show grains in the shades of grey. The shapes of grains have been well rendered by the program, and the growth of grains looks realistic. 6-fold densification of cells and reduction of color diversity allows to get results closer to reality, but it is a memory-intensive solution.





Picture 3.1. From the top: grain growth from the application, realistic steel grain growth [1]



Picture 3.2. From the left: realistic photo [2], grains with inclusions from the application (6-fold densification), grains in more realistic colour(6-fold densification)

4. Literature

1. R. C. Chena, C. Honga, J. J. Lia, Z. Z. Zhenga, P. C. Lib - *Austenite grain growth and grain size distribution in isothermal heat-treatment of 300M steel*
2. E. Khzouz - *Grain Growth Kinetics in Steels*
3. J.Pacyna - *Metaloznawstwo*
4. Tutorialspoint – Spring Web MVC Framework *https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm?fbclid=IwAR2gx2ucEcs7VDnmp2krYK0YAQHmhifwPr_Z5DpDDMFy7arnj10RYN3s1WI*
5. Official Site of Spring *http://spring.io/*
6. Official Site of Thymeleaf *https://www.thymeleaf.org*