| Multiscale modeling | | |
|---|---|---|
| Name Surname:<br>**Jan Pikulski** | Second report | Date:<br>**19.12.2019** |
| Index number:<br>**286220** | | Grade: |

## 1. Introduction

The main goal of this project was to expand the previously written program, by adding the possibility of generating various material microstructures, using the Monte Carlo method. Thanks to previously implemented solutions, this task was only composed of adding new functionalities to an already existing application.

The technological stack remained the same as previously and there was no need to add new technologies to the whole solution.

## 2. Added functionalities

a) The first new feature added to the GUI is the Generation Method dropdown, which allows the user to choose between the CA and Monte Carlo methods of microstructure generation. If no value is selected, system defaults to Cellular Automata. *Figure 1* shows the newly refreshed GUI and an example microstructure generated by the application for 2 states, using the Monte Carlo method.

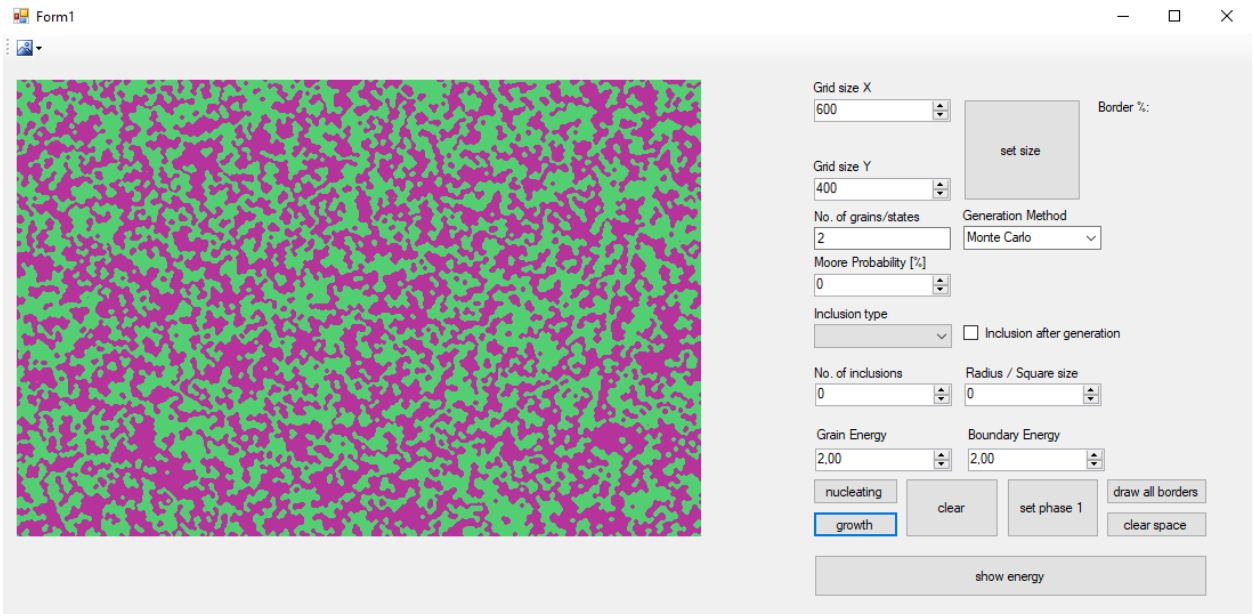


*Figure 1* – Refreshed GUI with example output generated for 2 grain states

b) In addition, the process of generating Dual phase microstructure was updated, so that now user can interchange both CA and Monte Carlo methods while generating specific grains for a phase. *Figure 2* shows a dual phase microstructure, where the first phase was generated with the Monte Carlo method for 2 states and the second phase was generated using Cellular Automata with 50 grains.

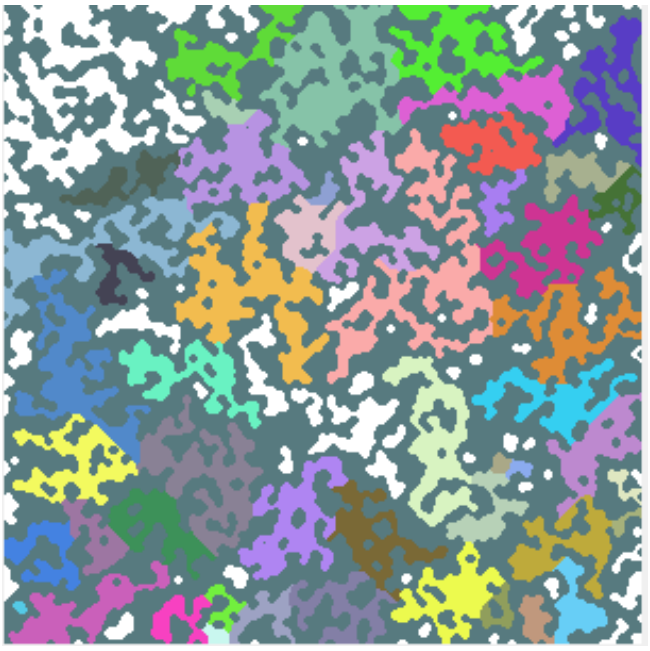

*Figure 2* – Dual phase microstructure MC -> CA

c) The last part which was implemented in this application was the process of distribution and visualization of energy in the microstructure. In order to allow the user to adjust specific values of energy (inside the grain and on the grain boundary separately), two additional numeric fields were added: **Grain Energy** and **Boundary Energy.** When both values are equal, the energy is distributed homogeneously, and each grain can be visualized using a different shade of blue in the viewport. When the numeric values of energy are different, energy is distributed heterogeneously, and grains which are located on the boundary are visualized with a light green color. In order to display current distribution of energy to the user, **show energy** button must be clicked. The two different types of energy visualization can be seen on *Figure 3*.

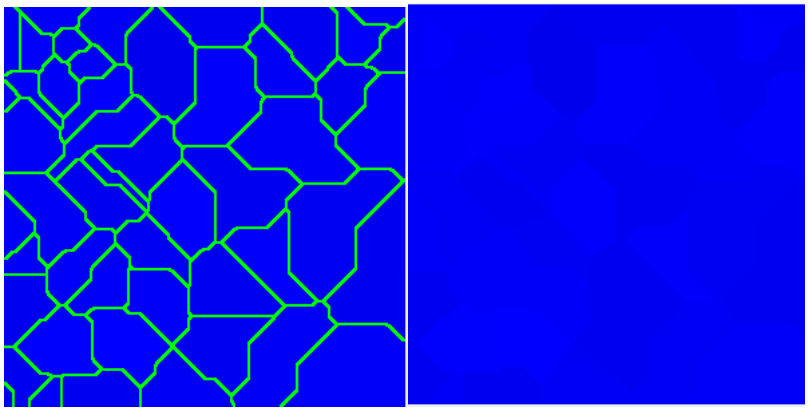

*Figure 3* – Heterogeneous (left) and Homogeneous (right) distribution of energy

## 3. Comparison and Conclusions

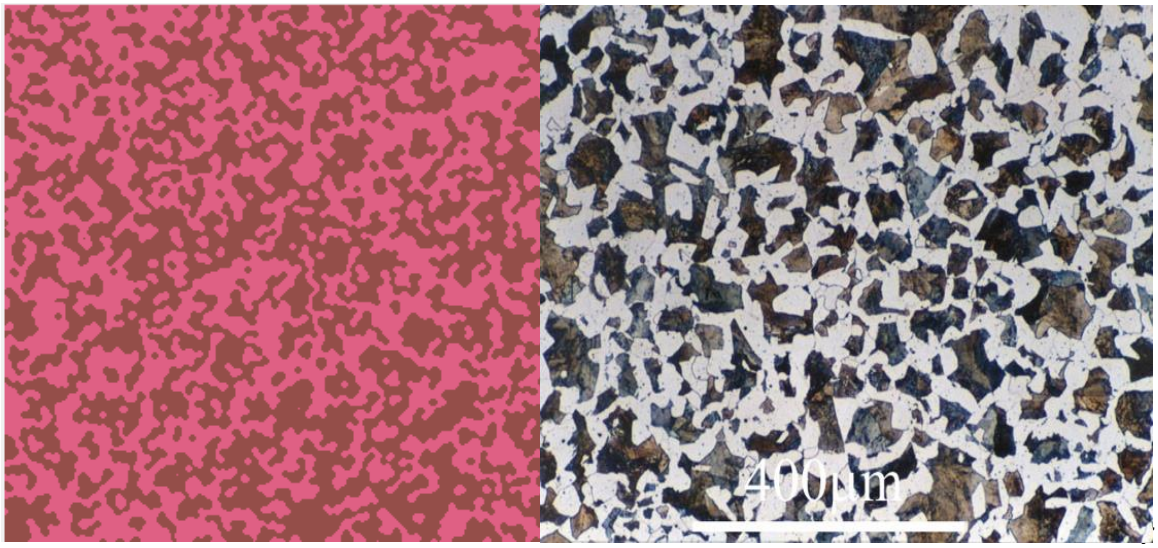


*Figure 4* – Monte Carlo generation output

*Figure 5* – Microstructure of Fe-0.4C steel

On *Figure 4* we can see a microstructure, which was generated with the Monte Carlo method for two states. *Figure 5* shows the microstructure of a piece of Fe-0.4C steel under a microscope.

As we can see, the output of the application is not a perfect representation of the picture shown on the right. None the less, by analyzing closely the shape of each grain, we are able to spot many similarities. The main advantage of the Monte Carlo method over Cellular Automata is that it is capable of generating round shapes, without any modifications to the algorithm. It is also very advantageous, that this method can create multiple regions, where one very small grain of one state is "locked" inside an enormous by comparison grain of a different state. As we can see, Monte Carlo offers results, which have some very interesting features that could never be achieved using only Cellular Automata.

Despite the fact, that generated output is close to reality, there are some factors, which make it impossible for the algorithm to work as efficiently as we would like it to work. The first thing that comes to mind is that we, as a user, have no control over the number of generated grains. User only provides the number of states which he wants to assign at random to each grain.
Another factor is the computational load of the algorithm. Counting energy of a cell, assigning to each cell at random (but only once per iteration!) a value of state, counting energy for the newly assigned state. Even with some optimization (assigning random state based on the neighborhood and not on the whole list of states, selecting random pixels only on grain boundaries) the program still is very slow. A refactor of the code in order to

make it capable of thread computing and utilizing the GPU more than the CPU could improve the overall performance of the entire application.

The technological stack used in this project once again proved to be an amazing tool, capable of creating functional and accurate code. However, due to the problematic nature of the Monte Carlo algorithm, a significant increase in program execution time was registered, in comparison to Cellular Automata (for a simple 300x300 array, the Monte Carlo method can run about 2-3 times slower than CA.

Overall, the application meets all the requirements established at the beginning of this project, up until the point of energy visualization. After that, all other points regarding static recrystallization were not implemented, due to a strict time constraint and conflict with other projects.