

Multiscale modeling – Application implementation report.

1. Technology stack review.

During the planning phase of application development, it was decided to make implementation in Java Standard Edition – high-level object-oriented programming language natively supporting multithreading. Moreover, to simplify the building process and dependencies management, Maven build automation tool was also used – proper rules were defined to easily create standalone jar executable on every platform. To address the visualization requirement, it was decided to use JavaFX with the usage of the Scene Builder tool which helps to design a graphical user interface view using the drag and drop method.

In the project, two external libraries dependency was defined: *org.apache.commons* – a package of Java utility classes and *JUnit* – library for unit testing.

Implementation was made using a continuous integration method using the git version control system and remote repository on the github.com website. The main view of the project's remote repository was presented in figure 1.

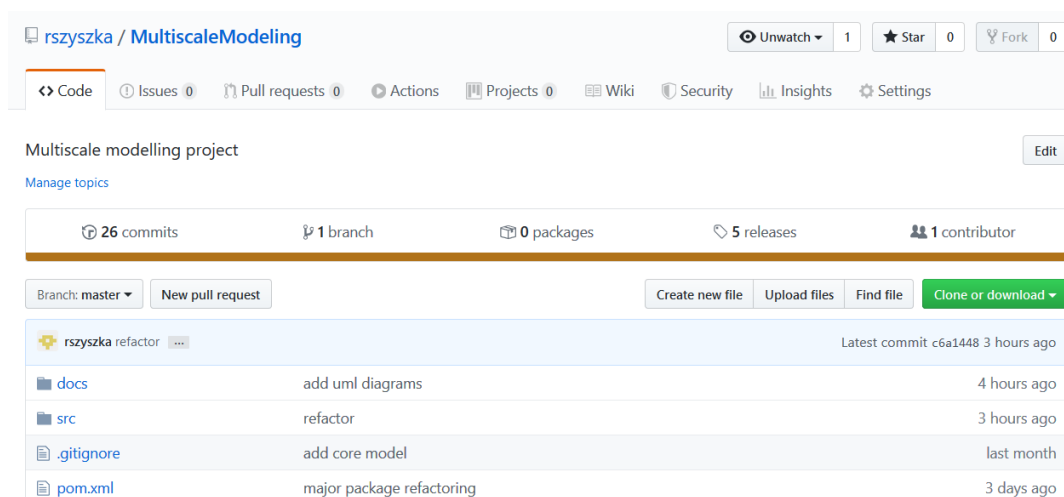
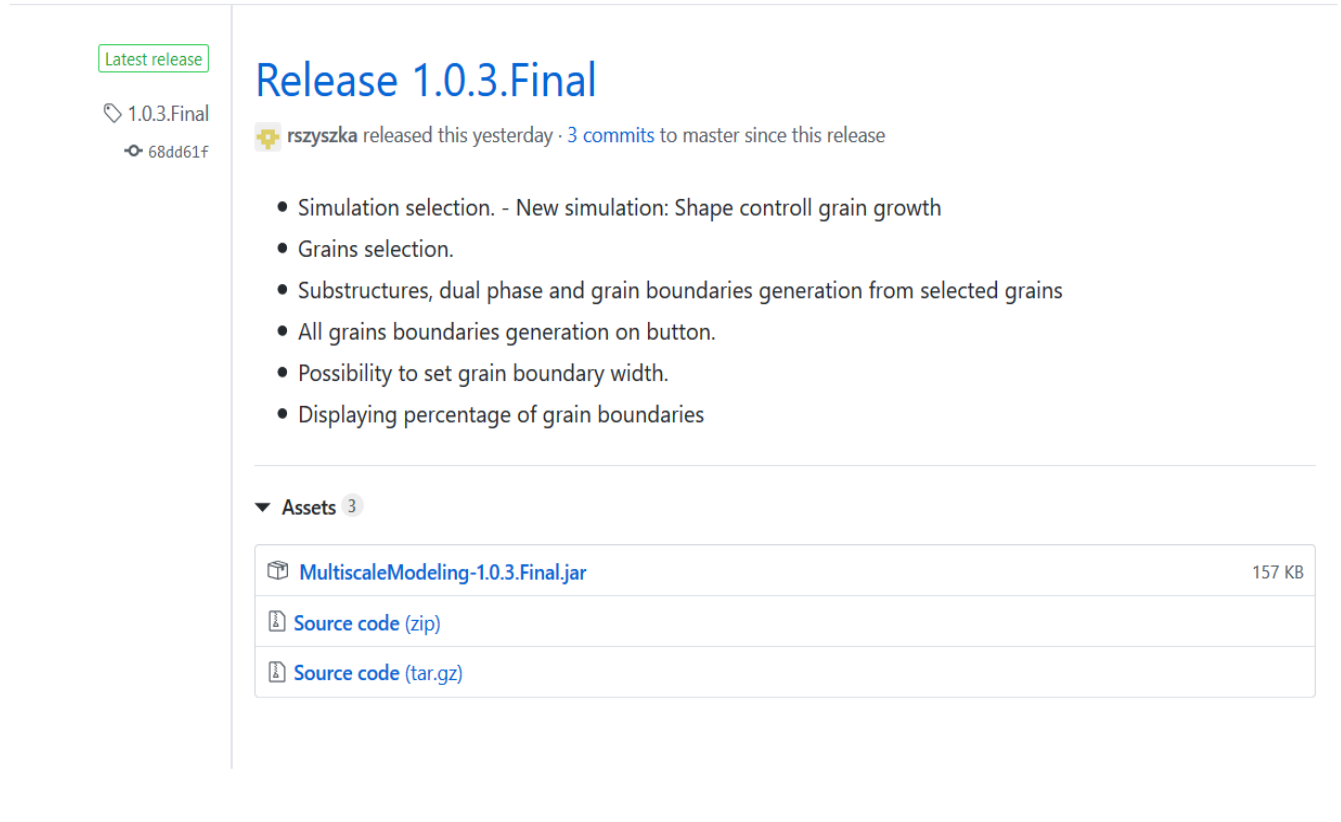


Figure 1. Project repository on github.com

As soon as the milestone was reached, the next version of the application was released with a standalone jar executable file attached to make possible fast downloading and running a stable version of the application. An example of a release was shown in figure 2.



The screenshot shows the GitHub release page for the repository 'MultiscaleModeling'. The release is titled 'Release 1.0.3.Final' and is marked as the 'Latest release'. It was released by 'rszyska' yesterday, with 3 commits since the previous release. The release includes a list of features: Simulation selection (new simulation: Shape controll grain growth), Grains selection, Substructures, dual phase and grain boundaries generation from selected grains, All grains boundaries generation on button, Possibility to set grain boundary width, and Displaying percentage of grain boundaries. Below the list, there is a section for 'Assets' with 3 items: 'MultiscaleModeling-1.0.3.Final.jar' (157 KB), 'Source code (zip)', and 'Source code (tar.gz)'.

Latest release

1.0.3.Final
68dd61f

Release 1.0.3.Final

rszyska released this yesterday · 3 commits to master since this release

- Simulation selection. - New simulation: Shape controll grain growth
- Grains selection.
- Substructures, dual phase and grain boundaries generation from selected grains
- All grains boundaries generation on button.
- Possibility to set grain boundary width.
- Displaying percentage of grain boundaries

▼ Assets 3




 MultiscaleModeling-1.0.3.Final.jar	157 KB
 Source code (zip)	
 Source code (tar.gz)	

Figure 2. Latest release view on github.com

For anyone interested in details of the project repository content it is available publicly via URL:

<https://github.com/rszyska/MultiscaleModeling>

2. Implementation specification.

The general concept of the application's architecture was to use a Model-View-Controller pattern which separates an application into three main logical components. Each of these components handles specific development aspects of an application. To fulfill the initial concept, the package structure was properly designed (figure 3).

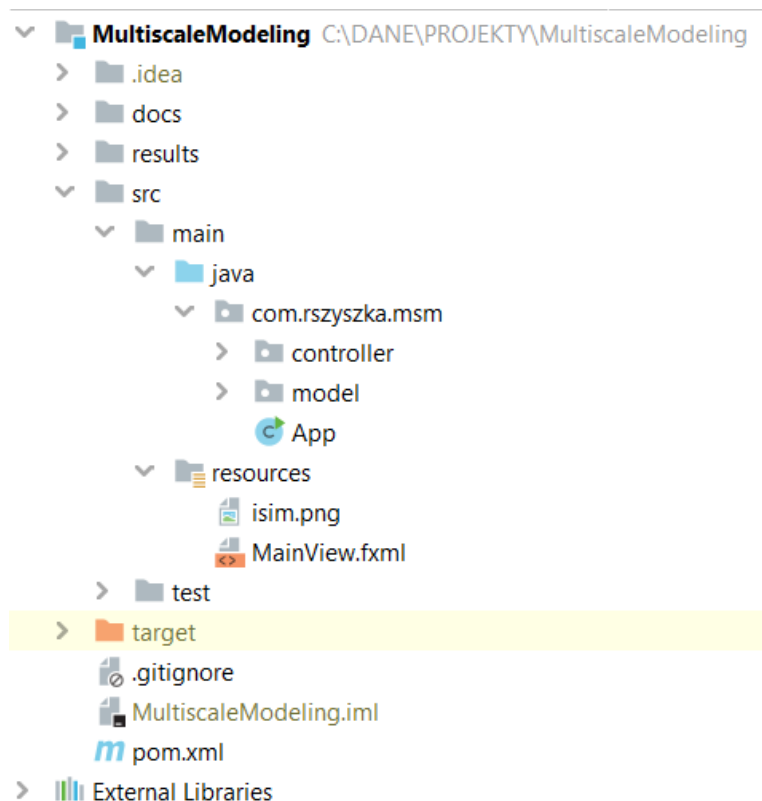
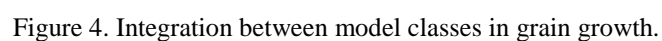


Figure 3. Package structure.

In the resources directory application's main view FXML file is placed in which look of GUI and controls are defined. The controller package contains the main view controller class which is some kind of adapter between all model classes data and application view.



On the diagram, one can notice that inheritance was used to implement shape control grain growth algorithm. It allowed to strongly reduce code duplication.

To keep GUI responsive during grain growth simulation it was delegated to run in a separate thread. Communication between growth thread and GUI thread is a trivial implementation of the Observer Pattern. Whereas, proper grain growth algorithm implementation is passed based on context. This was achieved by Strategy Pattern. Figure 5 presents a UML Diagram of the described situation.

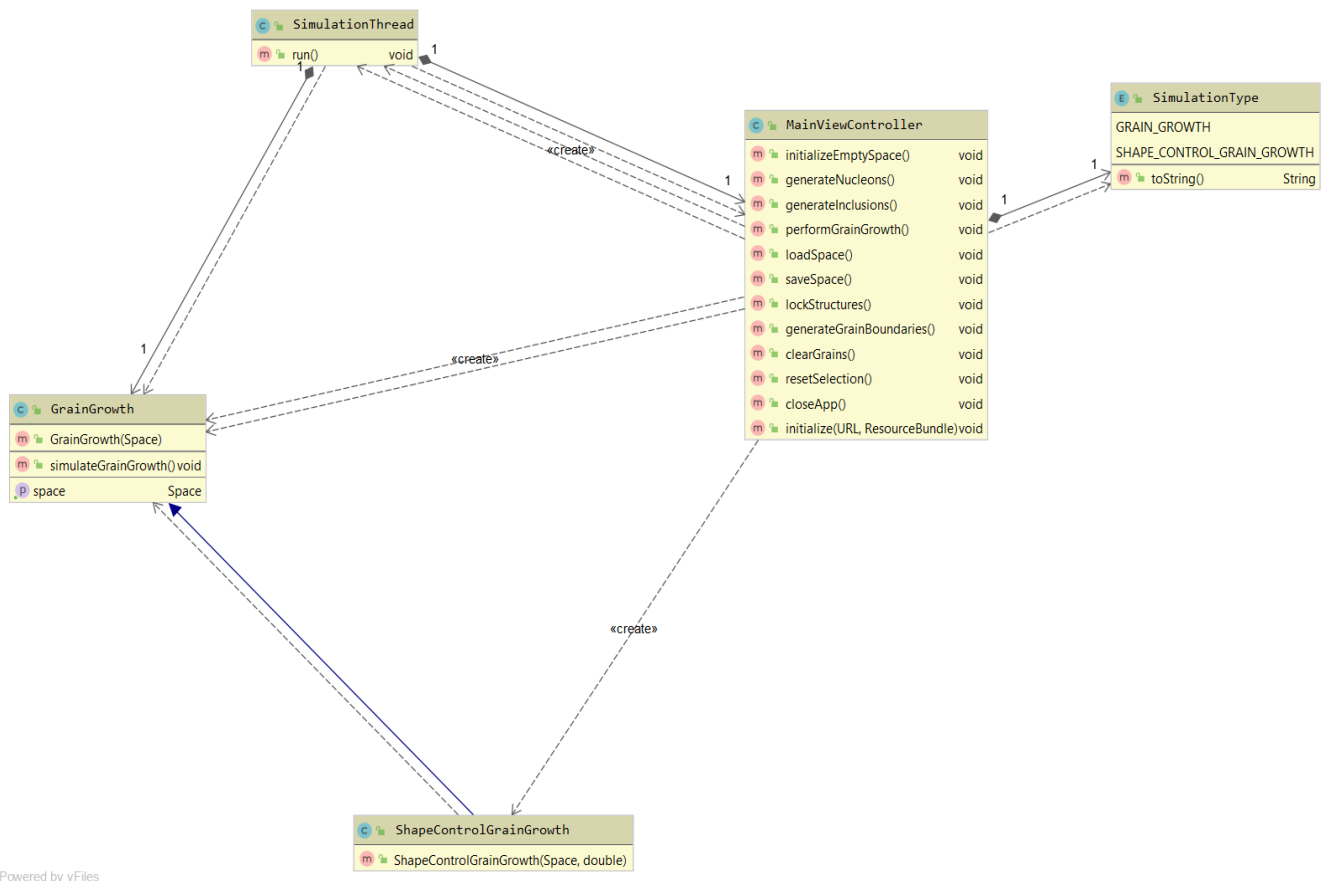


Figure 5. Controller integration with simulation thread and grain growth models.

Classes responsible for generating inclusions, substructures, dual-phase and grain boundaries and dependencies between them were presented on the UML diagram in figure 6.

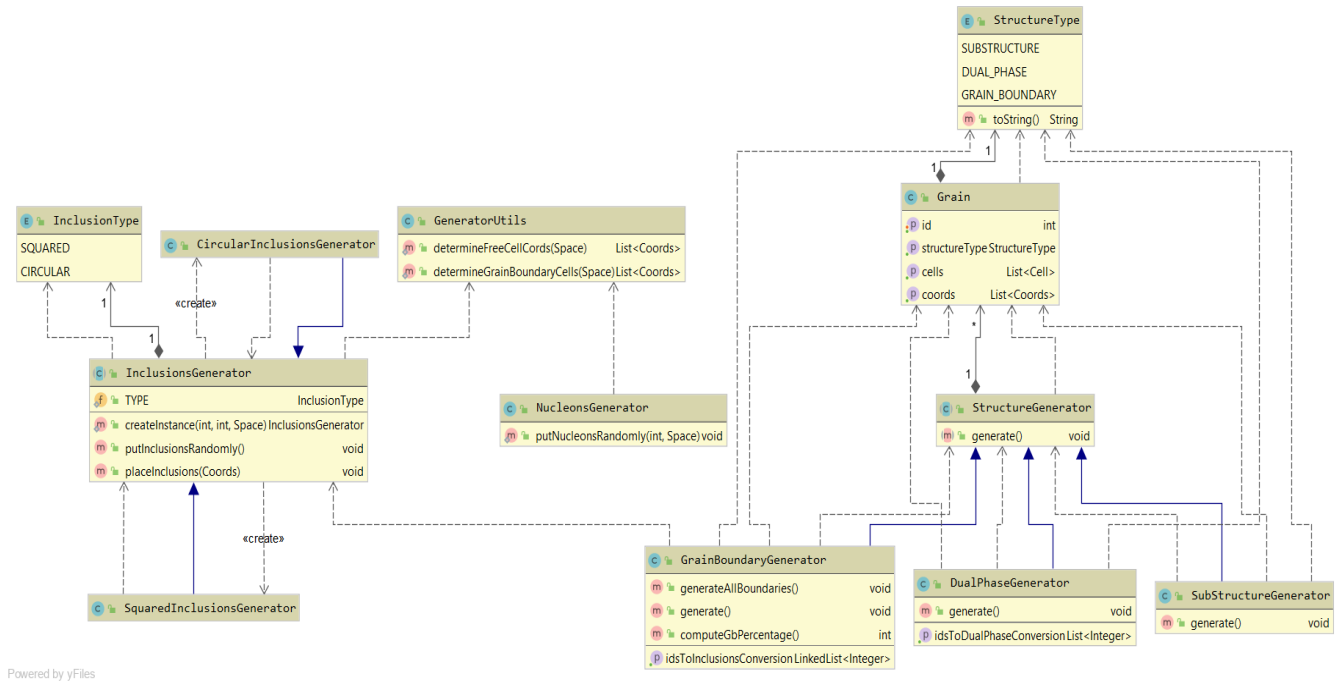


Figure 7. Generators classes UML Diagram.

Since all structure generation classes work on grains, the corresponding class was implemented which aggregates proper cells and their coordinates. To meet the thickness of the grain boundaries requirement, the functionality of the *InclusionsGenerator* class was used to draw inclusions of given size along the grain boundary.

3. Application user interface.

The main view of the application's GUI is shown in figure 8.

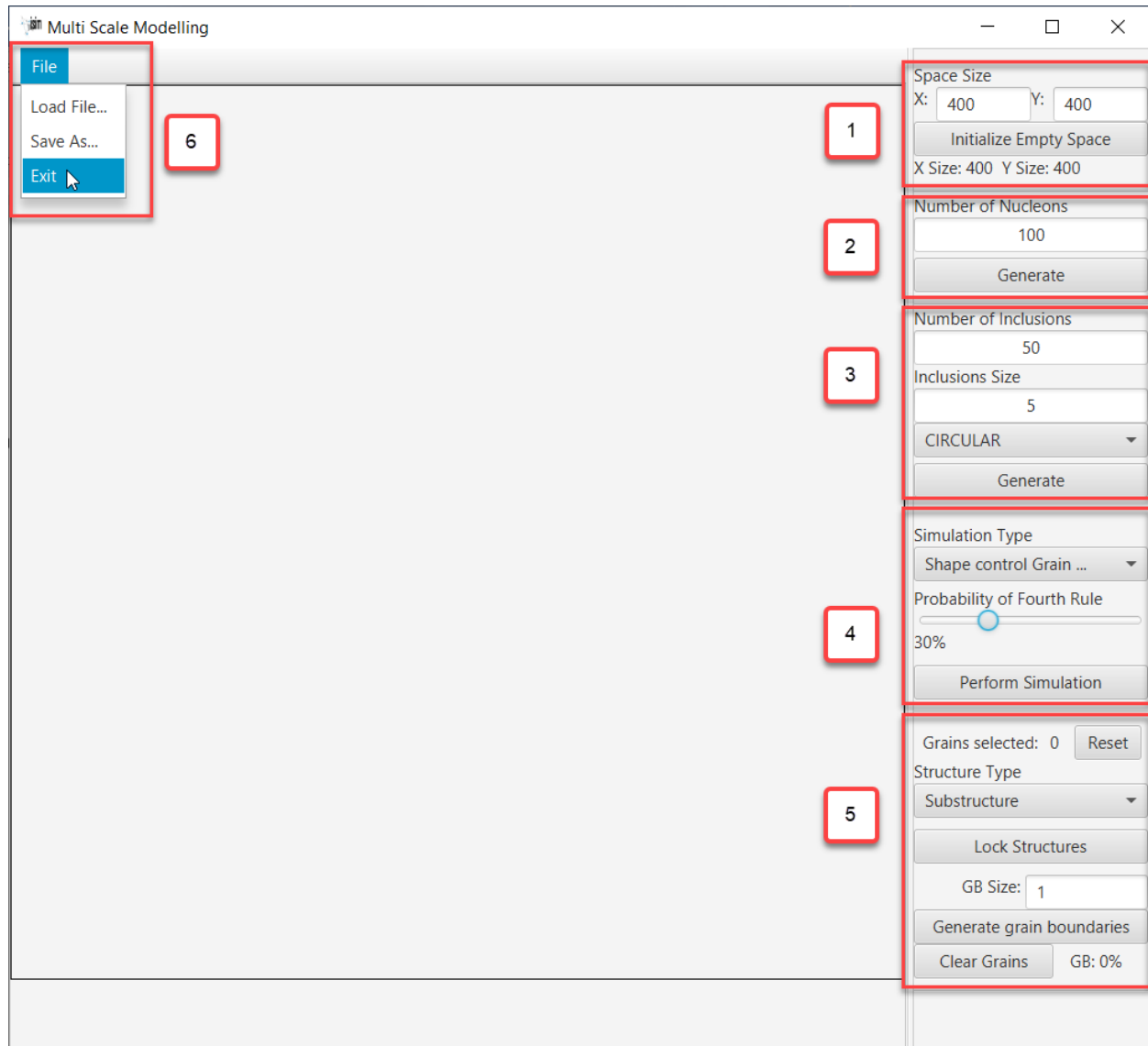


Figure 8. Application main view.

- In section no 1. the user can specify size and initialize empty space. In small labels below button is displayed current grid size.
- In section no 2. user is allowed to generate a given number of nucleons.

- Section no 3. is used to generate inclusions. The user can specify amount, size and type of inclusions.
- Section no 4. is a place where the user can choose simulation type and if it's *Shape Control Grain Growth* additionally the probability of the fourth rule can be adjusted.
- In section no 5. the user can perform selected grains transformation into selected structures. If the user wants to transform selected grains into grain boundaries also thickness can be specified. Moreover, all grains boundaries can be generated on button action. Additionally, grain boundaries occupation ratio is displayed and the amount of currently selected grains. The *Reset* button unselects all grains.
- Section no 6. is a *File* context menu from which the user can either load or save cellular automata space or close application.

All text fields on GUI have validation implemented to forgive user's mistakes.

4. Implemented application functionalities.

Grain growth simulations.

The main functionality is performing grain growth simulations. As previously mentioned, the user can choose between two simulation types: *Grain Growth* or *Shape Control Grain Growth*. In figure 9 results of both simulations were shown.

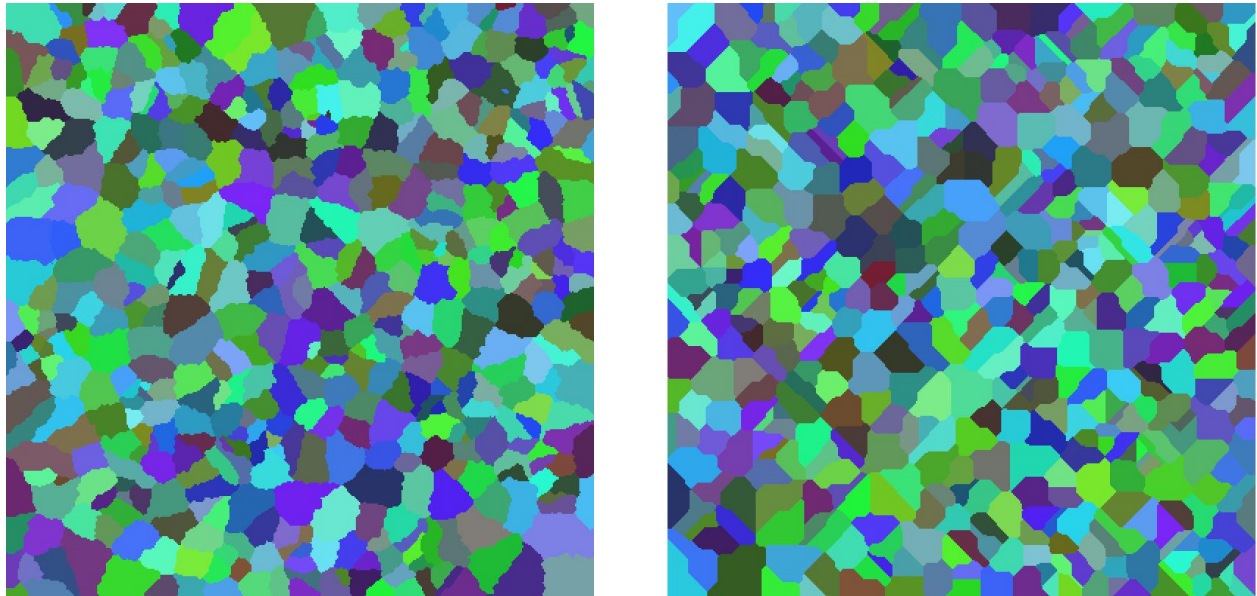


Figure 9. Microstructure generated using *Shape Control Grain Growth* (left) and *Simple Grain Growth* (right).

Inclusions generation

Inclusions can be generated before growth – in that case, they are placed randomly in empty cells in space or after growth – then inclusions are generated randomly in grain boundaries. The user can make the choice between two shape types of inclusions: squared or circular. Moreover, their size can be adjusted. The result of generation 20 circular inclusions before simulation and then after growing 50 nucleons and subsequent generation of 20 squared inclusions is presented in figure 10.

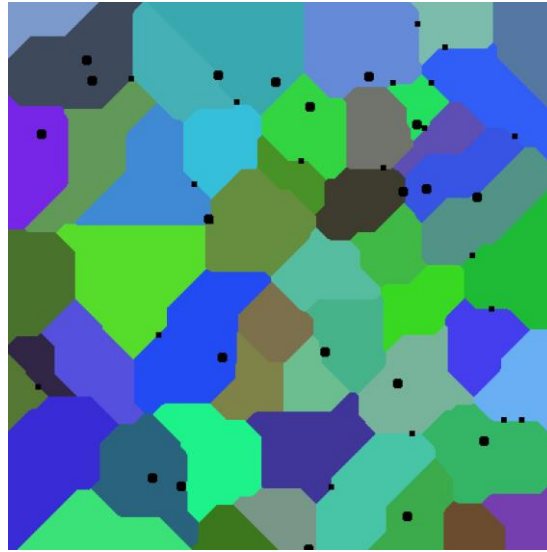


Figure 10. Microstructure with inclusions generated before (circular) and after grain growth (squared).

Selected grains transformations

The user can perform selected grains transformation into three structure types: *Substructures*, *Dual-Phase* or *Grain Boundaries*. Selected grains become highlighted in red. The user can freely select and unselect grains and change the type of desired grain structure during selection. If all grains have been selected user can call *Lock structures* button action on which picked grains will transform into selected structures and the rest of the space will be cleared. The described functionality is shown in figure 11. After clearing space new nucleons can be placed and growth while locked grains remain unchanged.

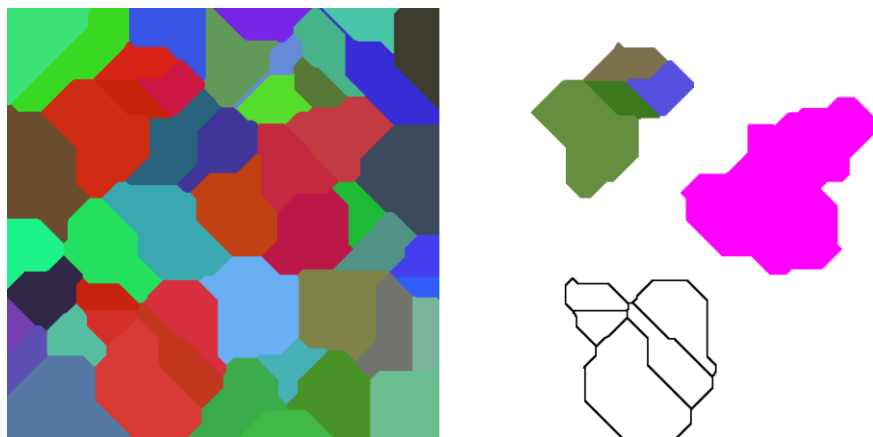


Figure 11. Grains selected to transformation to substructures, dual-phase and grain boundary simultaneously.

Before (left) and after locking and clearing the remaining space (right).

Generating all grain boundaries.

The user can call *Generate grain boundaries* button action on which all grain boundaries will be drawn. After that proper percentage ratio will be displayed and if the user wants to *Clear Grains* button action can be called to leave only grain boundaries on space. Described functionality is shown in figures 12 and 13.

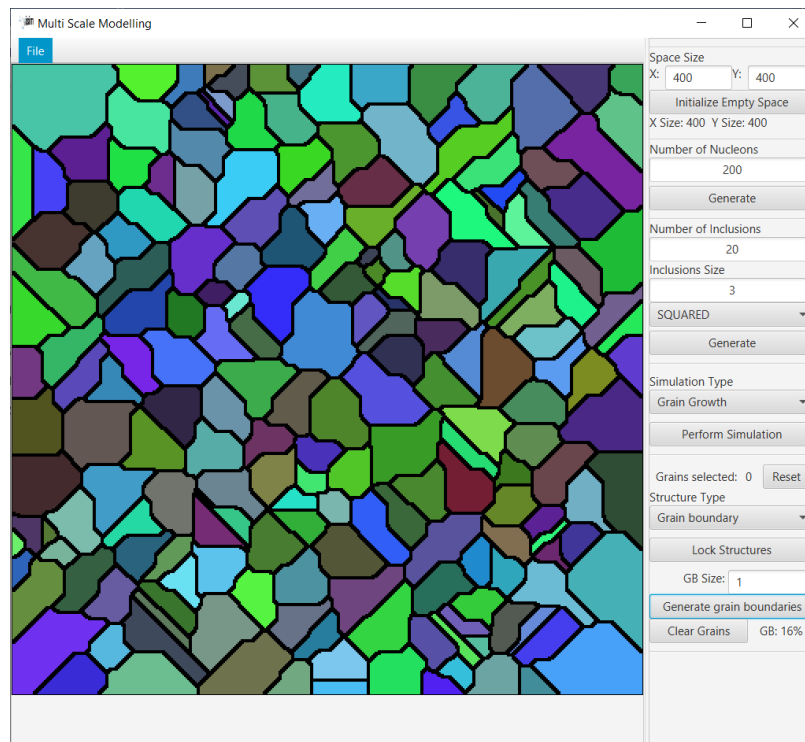


Figure 12. Application view with all the grain boundaries generated.

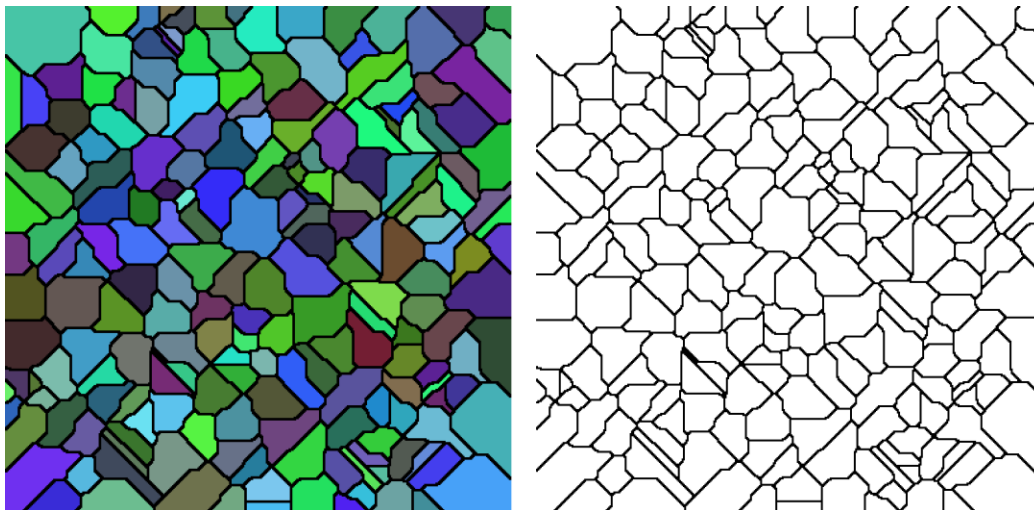


Figure 13. Grain boundaries before (left) and after (right) clearing grains.

Saving and loading space.

The user has the possibility to save space in any step of his work. Possible formats are png or txt file. Microstructures saved that way can be loaded later into the application to continue earlier disturbed work. Figure 14 shows saving space dialog whereas figure 15 shows an example of a saved text file.

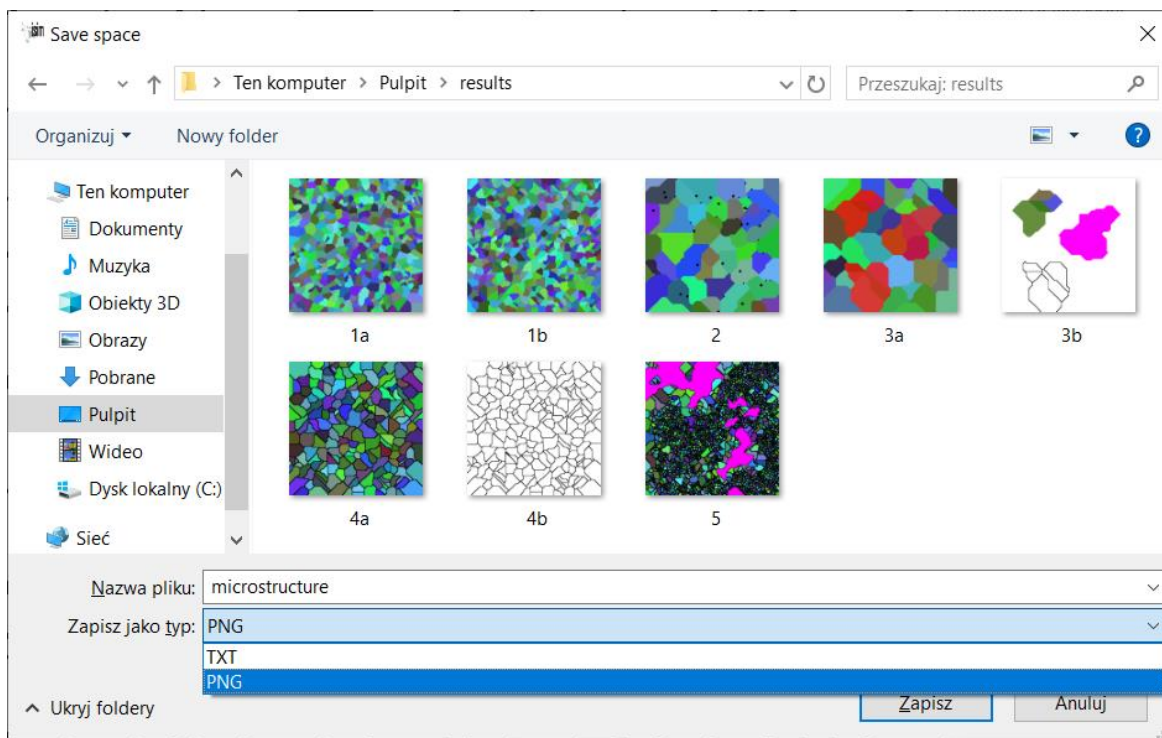


Figure 14. Saving space dialog.

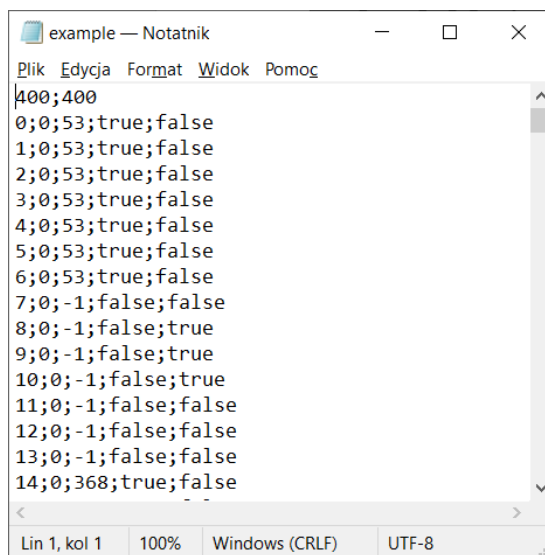


Figure 15. Space saved to a text file. The first line is space size. In next lines, properties of all cells are saved (x; y; id; isGrowable; isGrainBoundary)

5. Reconstruction of real microstructures

To show possibilities of the application an attempt of reconstructing a real microstructure was taken. The first microstructure was taken from UltraHigh Carbon Steel datasets. In below micrograph primary microconstituents are martensite and/or bainite [1]. In figure 16 comparison of real micrograph and generated microstructure was presented.

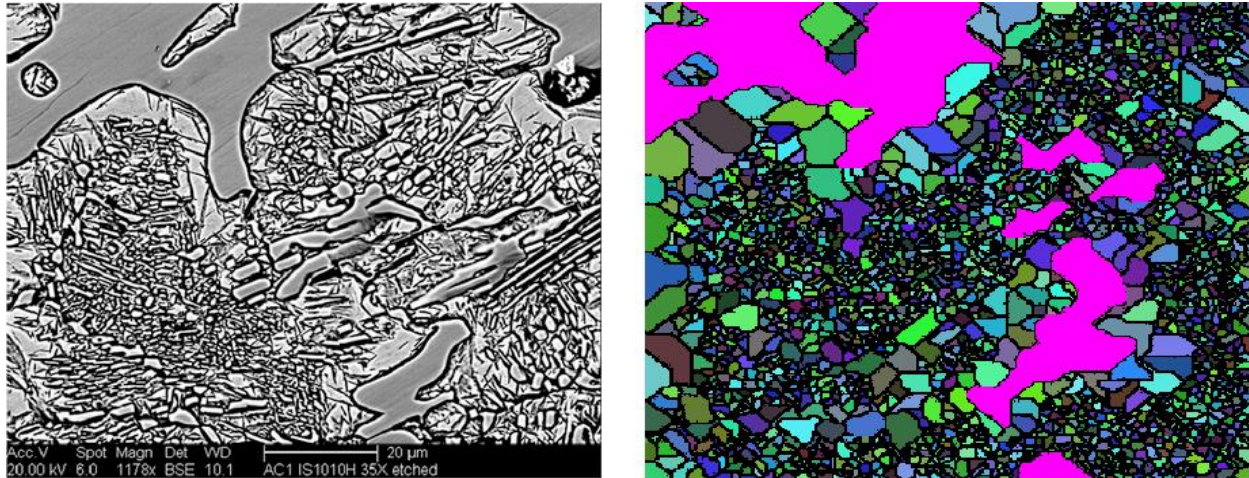


Figure 16. Comparison real (left) to generated (right) carbon steel microstructure.

The second microstructure taken to reconstruction is steel AC810 [2]. The result obtained is shown in figure 17.

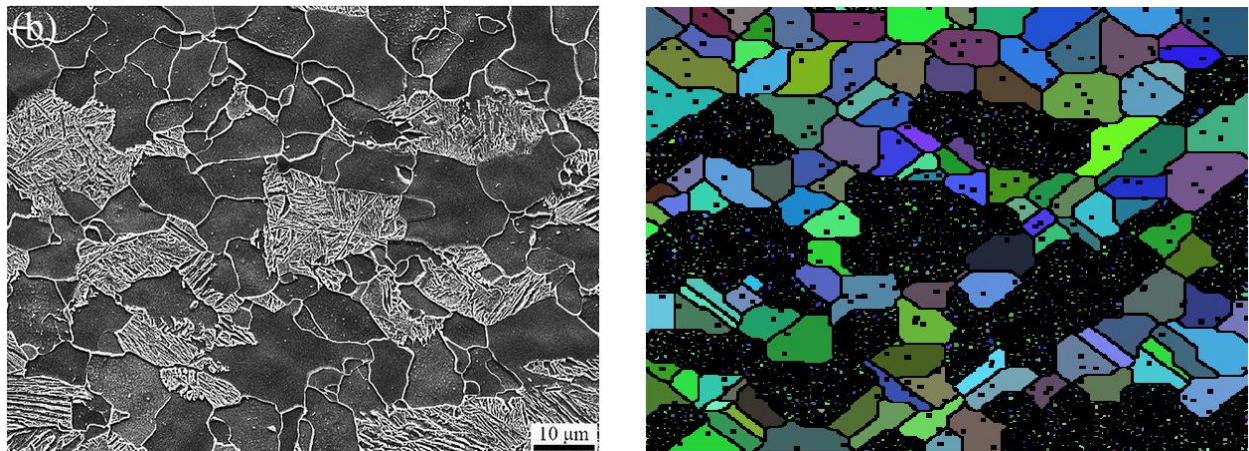


Figure 16. Comparison real (left) to generated (right) AC810 steel microstructure.

In the presented comparisons, there can be noticed a lot of similarities between real and generated microstructures. Gained results showed that in the use of developed application there is a possibility of generating microstructures with promising accuracy.

6. Bibliography

- [1] Brian L. DeCost, Toby Francis, Elizabeth A. Holm, Exploring the microstructure manifold: Image texture representations applied to ultrahigh carbon steel microstructures, *Acta Materialia*, Volume 133, 2017, Pages 30-40, ISSN 1359-6454.

(<http://www.sciencedirect.com/science/article/pii/S1359645417303919>)

- [2] M.Y. Sun, X.L. Wang, Z.Q. Wang, X.M. Wang, X.C. Li, L. Yan, R.D.K. Misra, The critical impact of intercritical deformation on variant pairing of bainite/martensite in dual-phase steels, *Materials Science and Engineering: A*, 2019, 138668, ISSN 0921-5093.

(<http://www.sciencedirect.com/science/article/pii/S0921509319314546>)