Marcin Szumlański, IS stacjonarne

Multiscale Modelling, 1st report

Goal of 1st part of assignment was to implement a grain growth application using cellular automata as foundation. My application uses Von Neumann neighborhood where cells are expanding in a cross-style manner. Cells area is 300x300 with absorbing boundary conditions, meaning the outermost cells on the edges are fixed with specific state (empty in my case).

Technology:

Java

Java is a general-purpose computer-programming language that is class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

I chose Java due to its ease of use, out-of-the box graphical user interface support (JavaSwing) and portability. The development time was also important factor.

GUI

The GUI is divided into 2 parts: the foreground one which consists mainly of text boxes and couple buttons (fig. 1), and an expanding menu (fig. 2) which is used for exporting/importing purposes.

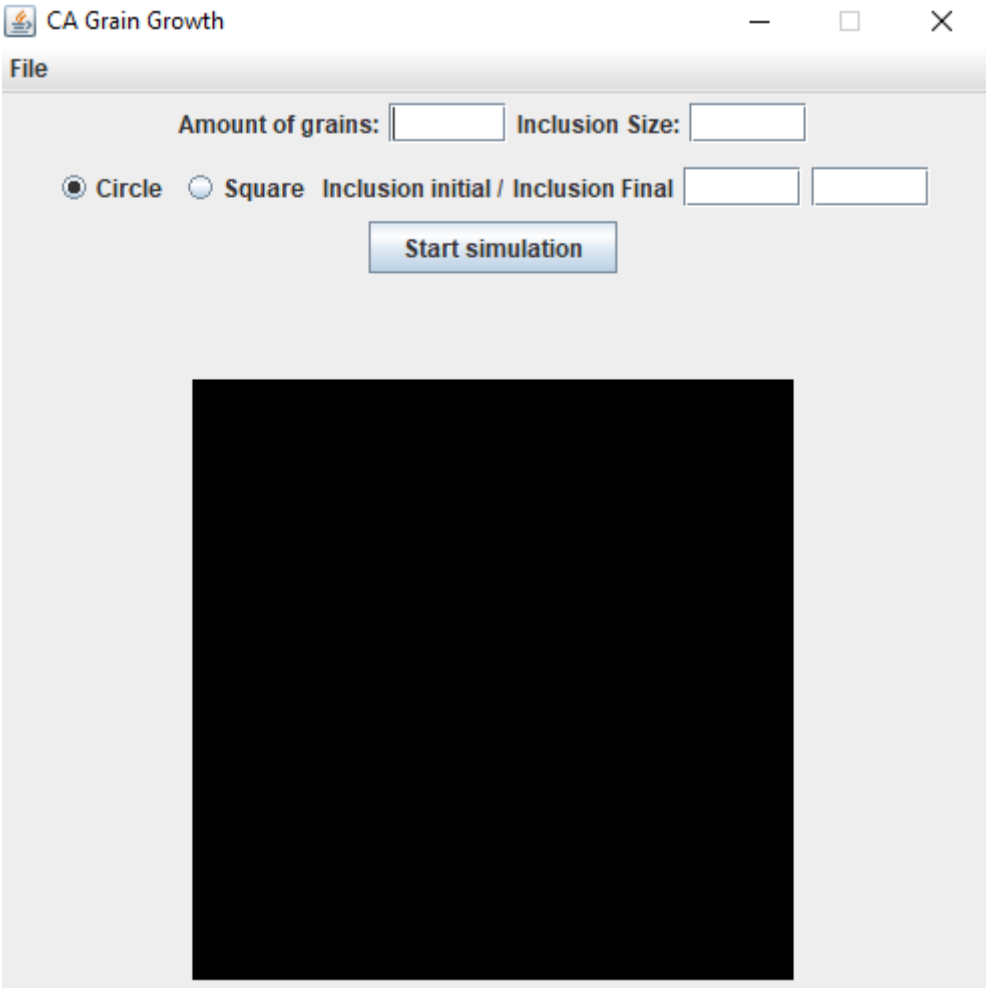


Fig. 1 Initial view of the application

Amount of grains: amount of grains to randomize from, each grain is assigned unique ID

Inclusion size: size of inclusion (1 cell is equal 1 size unit) – diameter for square ones and width for square inclusions

Circle and Square radio buttons: used which type of inclusion form to choose from

Inclusion Initial/Inclusion Final: amount of inclusions for both pre-simulation state and post-simulation state, the former ones being randomized over entire board, whereas for the latter, the coordinates of centers of inclusions are randomized over borders of grains.

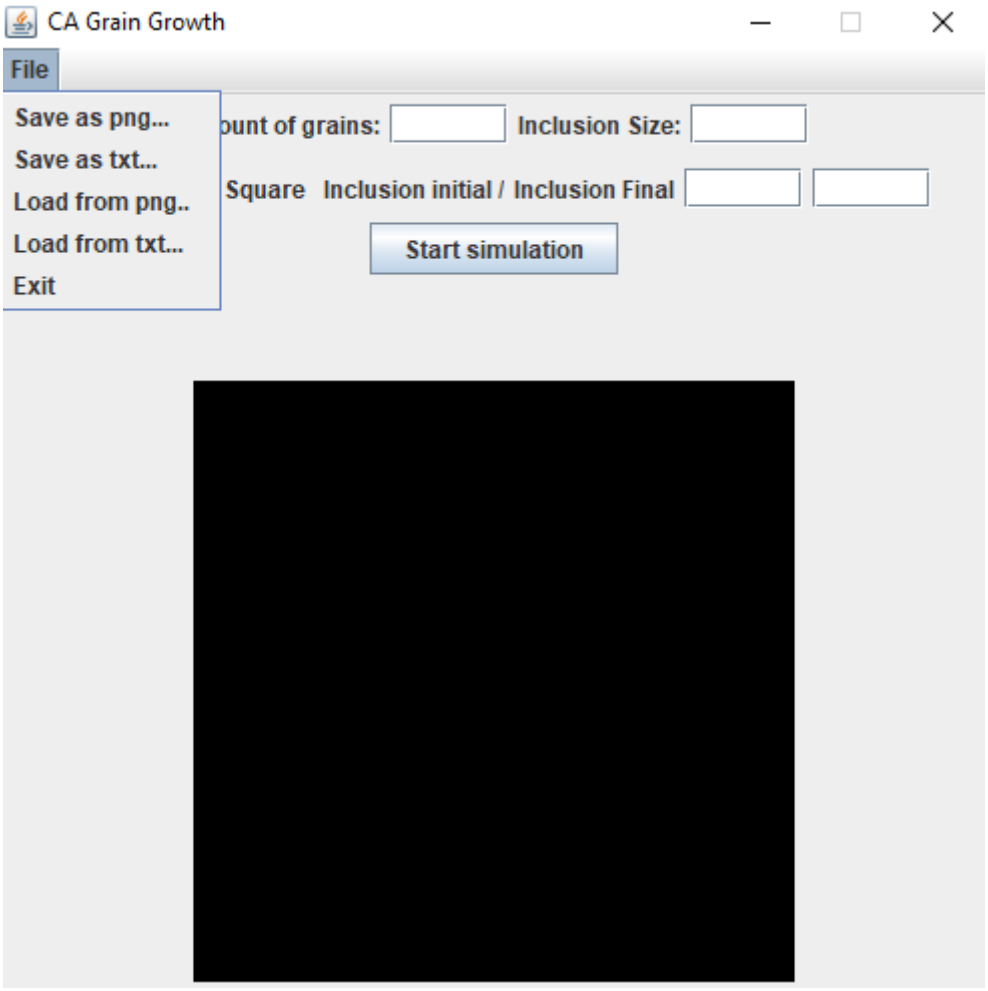


Fig. 2 Expanding menu used for import/export purposes

Import as PNG&txt : Used for importing purposes as bitmap/text file, a directory menu pops up to choose localization where to import from

Export as PNG&txt : Used for exporting purposes as bitmap/text file, a directory menu pops up to choose localization where to export to

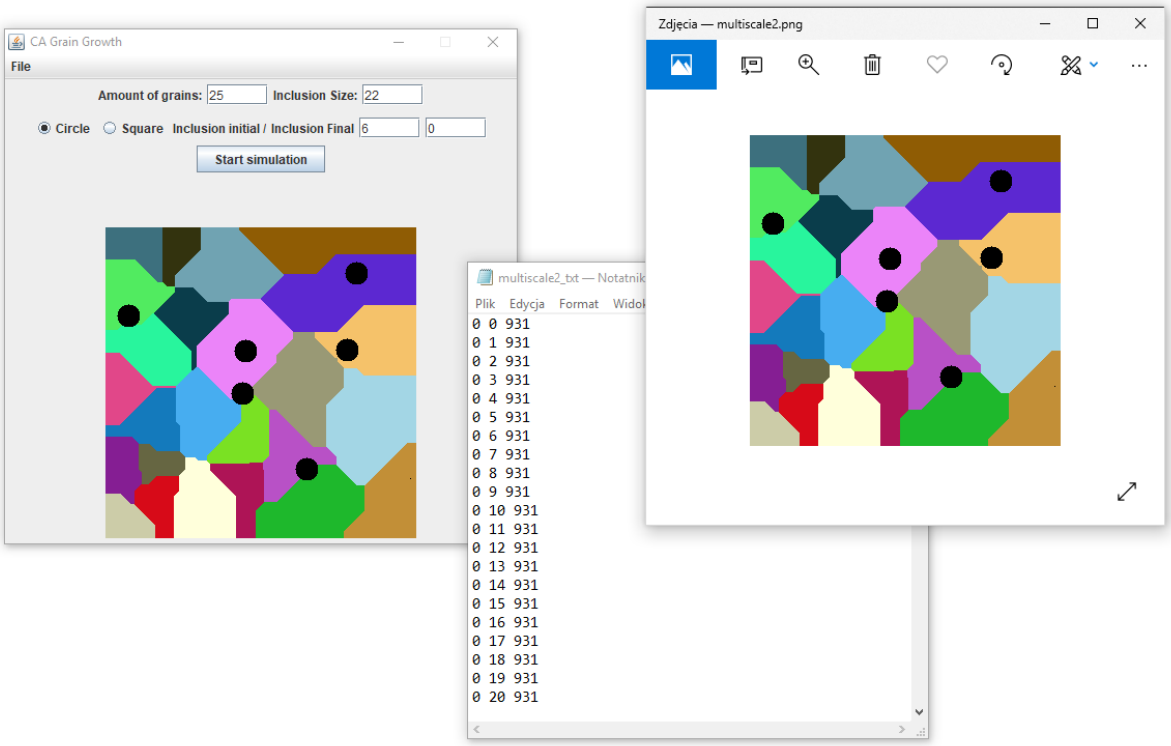
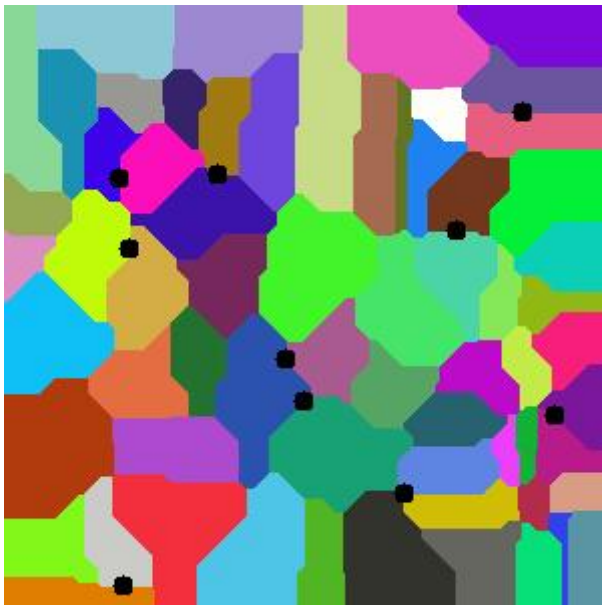


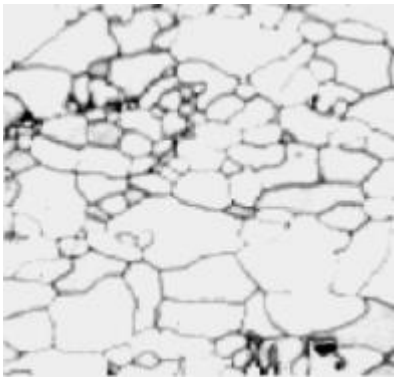
Fig 3.View of final state of application, with most features implemented

First three classes have been implemented fully(fig. 3), with simple grain growth using Von Neumann neighborhood and absorbing boundary conditions (300x300 space). Import/export to both txt and .png file formats and inclusion handling for both post- and pre-simulation conditions.

Comparison of real life vs simulation



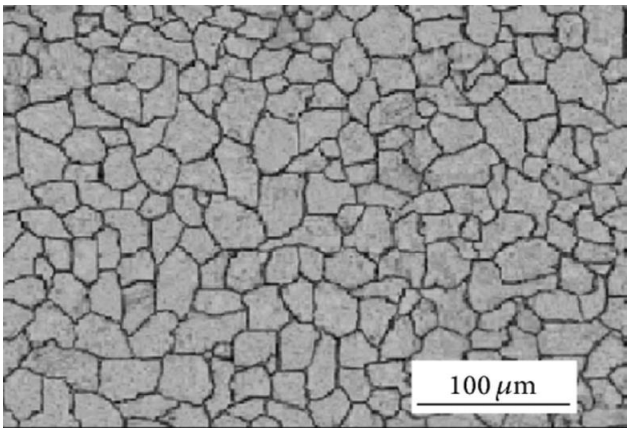
Simulation

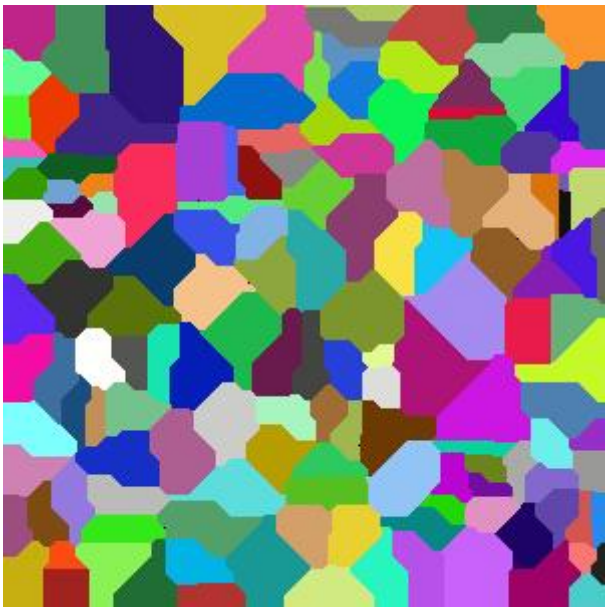


Actual photo

Fig. 4 Microstructure of 316LN stainless steel vs generated one with post-simulation inclusions

Obviously, the grains in real life are not as colorful as in simulation, but the shapes and overall feel is roughly the same. As are inclusions.





Simulation Actual photo

Fig. 5 Microstructure of 316LL austenistic stainless steel vs generated one with post-simulation

Yet another case this time without inclusions themselves but we can clearly see that the shape of grains is corresponding to real ones.

Summary

Most features required for classes have been implemented including grain growth with GUI, export/import to txt/png formats and inclusions handling. Java offers a variety of packages useful in such case like Java Swing framework, Bitmap class and many others. The generated microstructures do resemble their real life counterparts and look very much alike. Application runs fast on 300x300 space and its very easy to use, so it can be used for an actual science work regarding studying variety of materials and their characteristics.