Header File:

```
#pragma once

#include <cstdlib>

#include <string.h>

#include <time.h>

#include <cstdlib>

#include <fstream>

#include <vector>

#include <Windows.h>


namespace simulationMultiscaleModelling

{

	using namespace System;

	using namespace System::ComponentModel;

	using namespace System::Collections;

	using namespace System::Windows::Forms;

	using namespace System::Data;

	using namespace System::Drawing;

	using namespace System::IO;


	/// <summary>

	/// Summary for MyForm

	/// </summary>

	public ref class MyForm : public System::Windows::Forms::Form

	{

	public:

		MyForm(void)

		{

			InitializeComponent();

			//

			//TODO: Add the constructor code here

			//

		}

	protected:
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		~MyForm()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::Label^  label1;
	protected:
	private: System::Windows::Forms::Label^  label2;
	private: System::Windows::Forms::Label^  label3;
	private: System::Windows::Forms::Button^  startSymulationButton;
	private: System::Windows::Forms::TextBox^  xSizeValueTextBox;
	private: System::Windows::Forms::TextBox^  ySizeValueTextBox;
	private: System::Windows::Forms::MenuStrip^  menuStrip1;
	private: System::Windows::Forms::ToolStripMenuItem^  fileToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^  microstructureToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^  importToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^  exportToolStripMenuItem;
	private: System::Windows::Forms::DomainUpDown^  amoutNucleonDomianUpDown;
	private: System::Windows::Forms::Panel^  panel1;
	private: System::Windows::Forms::Label^  label4;
	private: System::Windows::Forms::Label^  label5;
	private: System::Windows::Forms::Label^  label6;
	private: System::Windows::Forms::TextBox^  amountOfInclusionsTextBox;
	private: System::Windows::Forms::TextBox^  sizeOfInclusionsTextBox;
	private: System::Windows::Forms::ComboBox^  typeOfInclusionComboBox;
	private: System::Windows::Forms::PictureBox^  pictureBox1;
	private: System::Windows::Forms::ToolStripMenuItem^  toTXTToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^  toBMPToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^  FromTXTToolStripMenuItem;
	private: System::Windows::Forms::ToolStripMenuItem^  fromBitmapToolStripMenuItem;
```

```
private: System::Windows::Forms::Button^  addAfterInclusions;
private: System::Windows::Forms::Button^  addBeforeInclusions;
private: System::Windows::Forms::ComboBox^  neighborhoodMethoodComboBox;

private: System::Windows::Forms::Label^  label7;
private: System::Windows::Forms::Label^  label8;
private: System::Windows::Forms::ComboBox^  structureComboBox;
private: System::Windows::Forms::Button^  structureGenerateButton;
private: System::Windows::Forms::Label^  label9;
private: System::Windows::Forms::ComboBox^  grainsSelectedComboBox;
private: System::Windows::Forms::Label^  label10;
private: System::Windows::Forms::TextBox^  gbSizeTextBox;

private: System::Windows::Forms::Label^  label11;
private: System::Windows::Forms::TextBox^  gbAmountTextBox;
private: System::Windows::Forms::Button^  generateGBButton;
private: System::Windows::Forms::Button^  clearSpaceButton;



private: System::Windows::Forms::Button^  clearAll;
private: System::Windows::Forms::Label^  label12;
private: System::Windows::Forms::DomainUpDown^  amountGrainDomainUpDown;









private:
	/// <summary>
	/// Required designer variable.
	/// </summary>
	System::ComponentModel::Container ^components;
```

```
#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

	this->label1 = (gcnew System::Windows::Forms::Label());

	this->label2 = (gcnew System::Windows::Forms::Label());

	this->label3 = (gcnew System::Windows::Forms::Label());

	this->startSymulationButton = (gcnew System::Windows::Forms::Button());

	this->xSizeValueTextBox = (gcnew System::Windows::Forms::TextBox());

	this->ySizeValueTextBox = (gcnew System::Windows::Forms::TextBox());

	this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());

	this->fileToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

	this->microstructureToolStripMenuItem = (gcnew
	System::Windows::Forms::ToolStripMenuItem());

	this->importToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

	this->FromTXTToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

	this->fromBitmapToolStripMenuItem = (gcnew
	System::Windows::Forms::ToolStripMenuItem());

	this->exportToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

	this->toTXTToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

	this->toBMPToolStripMenuItem = (gcnew System::Windows::Forms::ToolStripMenuItem());

	this->amoutNucleonDomianUpDown = (gcnew System::Windows::Forms::DomainUpDown());

	this->panel1 = (gcnew System::Windows::Forms::Panel());

	this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());

	this->label4 = (gcnew System::Windows::Forms::Label());

	this->label5 = (gcnew System::Windows::Forms::Label());

	this->label6 = (gcnew System::Windows::Forms::Label());

	this->amountOfInclusionsTextBox = (gcnew System::Windows::Forms::TextBox());

	this->sizeOfInclusionsTextBox = (gcnew System::Windows::Forms::TextBox());

	this->typeOfInclusionComboBox = (gcnew System::Windows::Forms::ComboBox());

	this->addAfterInclusions = (gcnew System::Windows::Forms::Button());

	this->addBeforeInclusions = (gcnew System::Windows::Forms::Button());

	this->neighborhoodMethoodComboBox = (gcnew System::Windows::Forms::ComboBox());
```

```
this->label7 = (gcnew System::Windows::Forms::Label());

this->label8 = (gcnew System::Windows::Forms::Label());

this->structureComboBox = (gcnew System::Windows::Forms::ComboBox());

this->structureGenerateButton = (gcnew System::Windows::Forms::Button());

this->label9 = (gcnew System::Windows::Forms::Label());

this->grainsSelectedComboBox = (gcnew System::Windows::Forms::ComboBox());

this->label10 = (gcnew System::Windows::Forms::Label());

this->gbSizeTextBox = (gcnew System::Windows::Forms::TextBox());

this->label11 = (gcnew System::Windows::Forms::Label());

this->gbAmountTextBox = (gcnew System::Windows::Forms::TextBox());

this->generateGBButton = (gcnew System::Windows::Forms::Button());

this->clearSpaceButton = (gcnew System::Windows::Forms::Button());

this->clearAll = (gcnew System::Windows::Forms::Button());

this->label12 = (gcnew System::Windows::Forms::Label());

this->amountGrainDomainUpDown = (gcnew System::Windows::Forms::DomainUpDown());

this->menuStrip1->SuspendLayout();

this->panel1->SuspendLayout();

(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))-
>BeginInit();

this->SuspendLayout();

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(12, 41);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(49, 17);

this->label1->TabIndex = 0;

this->label1->Text = L"x Size:";

this->label1->Click += gcnew System::EventHandler(this, &MyForm::label1_Click);

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(143, 41);

this->label2->Name = L"label2";
```

```
this->label2->Size = System::Drawing::Size(50, 17);

this->label2->TabIndex = 1;

this->label2->Text = L"y Size:";

this->label2->Click += gcnew System::EventHandler(this, &MyForm::label2_Click);

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(277, 41);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(107, 17);

this->label3->TabIndex = 2;

this->label3->Text = L"Nucleon amout:";

//

// startSymulationButton

//

this->startSymulationButton->BackColor = System::Drawing::SystemColors::ActiveBorder;

this->startSymulationButton->Cursor = System::Windows::Forms::Cursors::Hand;

this->startSymulationButton->Location = System::Drawing::Point(429, 397);

this->startSymulationButton->Name = L"startSymulationButton";

this->startSymulationButton->Size = System::Drawing::Size(140, 75);

this->startSymulationButton->TabIndex = 3;

this->startSymulationButton->Text = L"SIMULATION";

this->startSymulationButton->UseVisualStyleBackColor = false;

this->startSymulationButton->Click += gcnew System::EventHandler(this,
&MyForm::startSymulationButton_Click);

//

// xSizeValueTextBox

//

this->xSizeValueTextBox->Location = System::Drawing::Point(12, 63);

this->xSizeValueTextBox->Name = L"xSizeValueTextBox";

this->xSizeValueTextBox->Size = System::Drawing::Size(100, 22);

this->xSizeValueTextBox->TabIndex = 4;

this->xSizeValueTextBox->Text = L"300";

//

// ySizeValueTextBox
```

```
//

this->ySizeValueTextBox->Location = System::Drawing::Point(146, 63);

this->ySizeValueTextBox->Name = L"ySizeValueTextBox";

this->ySizeValueTextBox->Size = System::Drawing::Size(100, 22);

this->ySizeValueTextBox->TabIndex = 5;

this->ySizeValueTextBox->Text = L"300";

//

// menuStrip1

//

this->menuStrip1->ImageScalingSize = System::Drawing::Size(20, 20);

this->menuStrip1->Items->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(1) { this->fileToolStripMenuItem });

this->menuStrip1->Location = System::Drawing::Point(0, 0);

this->menuStrip1->Name = L"menuStrip1";

this->menuStrip1->Size = System::Drawing::Size(775, 28);

this->menuStrip1->TabIndex = 6;

this->menuStrip1->Text = L"menuStrip1";

//

// fileToolStripMenuItem

//

this->fileToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(1) { this->microstructureToolStripMenuItem });

this->fileToolStripMenuItem->Name = L"fileToolStripMenuItem";

this->fileToolStripMenuItem->Size = System::Drawing::Size(44, 24);

this->fileToolStripMenuItem->Text = L"File";

//

// microstructureToolStripMenuItem

//

this->microstructureToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(2) {

this->importToolStripMenuItem,

this->exportToolStripMenuItem

});

this->microstructureToolStripMenuItem->Name = L"microstructureToolStripMenuItem";

this->microstructureToolStripMenuItem->Size = System::Drawing::Size(179, 26);

this->microstructureToolStripMenuItem->Text = L"Microstructure";
```

```
//

// importToolStripMenuItem

//

this->importToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(2) {

		this->FromTXTToolStripMenuItem,

				this->fromBitmapToolStripMenuItem

});

this->importToolStripMenuItem->Name = L"importToolStripMenuItem";

this->importToolStripMenuItem->Size = System::Drawing::Size(129, 26);

this->importToolStripMenuItem->Text = L"Import";

//

// FromTXTToolStripMenuItem

//

this->FromTXTToolStripMenuItem->Name = L"FromTXTToolStripMenuItem";

this->FromTXTToolStripMenuItem->Size = System::Drawing::Size(170, 26);

this->FromTXTToolStripMenuItem->Text = L"From TXT";

this->FromTXTToolStripMenuItem->Click += gcnew System::EventHandler(this,
&MyForm::FromTXTToolStripMenuItem_Click);

//

// fromBitmapToolStripMenuItem

//

this->fromBitmapToolStripMenuItem->Name = L"fromBitmapToolStripMenuItem";

this->fromBitmapToolStripMenuItem->Size = System::Drawing::Size(170, 26);

this->fromBitmapToolStripMenuItem->Text = L"From Bitmap";

this->fromBitmapToolStripMenuItem->Click += gcnew System::EventHandler(this,
&MyForm::fromBitmapToolStripMenuItem_Click);

//

// exportToolStripMenuItem

//

this->exportToolStripMenuItem->DropDownItems->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(2) {

this->toTXTToolStripMenuItem,

this->toBMPToolStripMenuItem

});

this->exportToolStripMenuItem->Name = L"exportToolStripMenuItem";

this->exportToolStripMenuItem->Size = System::Drawing::Size(129, 26);
```

```
this->exportToolStripMenuItem->Text = L"Export";

//

// toTXTToolStripMenuItem

//

this->toTXTToolStripMenuItem->Name = L"toTXTToolStripMenuItem";

this->toTXTToolStripMenuItem->Size = System::Drawing::Size(134, 26);

this->toTXTToolStripMenuItem->Text = L"To TXT";

this->toTXTToolStripMenuItem->Click += gcnew System::EventHandler(this,
&MyForm::toTXTToolStripMenuItem_Click);

//

// toBMPToolStripMenuItem

//

this->toBMPToolStripMenuItem->Name = L"toBMPToolStripMenuItem";

this->toBMPToolStripMenuItem->Size = System::Drawing::Size(134, 26);

this->toBMPToolStripMenuItem->Text = L"To BMP";

this->toBMPToolStripMenuItem->Click += gcnew System::EventHandler(this,
&MyForm::toBMPToolStripMenuItem_Click);

//

// amoutNucleonDomianUpDown

//

this->amoutNucleonDomianUpDown->Items->Add(L"20");

this->amoutNucleonDomianUpDown->Items->Add(L"19");

this->amoutNucleonDomianUpDown->Items->Add(L"18");

this->amoutNucleonDomianUpDown->Items->Add(L"17");

this->amoutNucleonDomianUpDown->Items->Add(L"16");

this->amoutNucleonDomianUpDown->Items->Add(L"15");

this->amoutNucleonDomianUpDown->Items->Add(L"14");

this->amoutNucleonDomianUpDown->Items->Add(L"13");

this->amoutNucleonDomianUpDown->Items->Add(L"12");

this->amoutNucleonDomianUpDown->Items->Add(L"11");

this->amoutNucleonDomianUpDown->Items->Add(L"10");

this->amoutNucleonDomianUpDown->Items->Add(L"9");

this->amoutNucleonDomianUpDown->Items->Add(L"8");

this->amoutNucleonDomianUpDown->Items->Add(L"7");

this->amoutNucleonDomianUpDown->Items->Add(L"6");

this->amoutNucleonDomianUpDown->Items->Add(L"5");
```

```
this->amoutNucleonDomianUpDown->Items->Add(L"4");
this->amoutNucleonDomianUpDown->Items->Add(L"3");
this->amoutNucleonDomianUpDown->Items->Add(L"2");
this->amoutNucleonDomianUpDown->Items->Add(L"1");
this->amoutNucleonDomianUpDown->Location = System::Drawing::Point(280, 64);
this->amoutNucleonDomianUpDown->Name = L"amoutNucleonDomianUpDown";
this->amoutNucleonDomianUpDown->Size = System::Drawing::Size(120, 22);
this->amoutNucleonDomianUpDown->TabIndex = 7;
this->amoutNucleonDomianUpDown->Text = L"2";
//
// panel1
//
this->panel1->BackColor = System::Drawing::SystemColors::ControlLight;
this->panel1->Controls->Add(this->pictureBox1);
this->panel1->Location = System::Drawing::Point(12, 110);
this->panel1->Name = L"panel1";
this->panel1->Size = System::Drawing::Size(399, 368);
this->panel1->TabIndex = 8;
//
// pictureBox1
//
this->pictureBox1->Location = System::Drawing::Point(3, 3);
this->pictureBox1->Name = L"pictureBox1";
this->pictureBox1->Size = System::Drawing::Size(393, 365);
this->pictureBox1->TabIndex = 15;
this->pictureBox1->TabStop = false;
this->pictureBox1->Visible = false;
//
// label4
//
this->label4->AutoSize = true;
this->label4->Location = System::Drawing::Point(426, 110);
this->label4->Name = L"label4";
this->label4->Size = System::Drawing::Size(142, 17);
this->label4->TabIndex = 9;
```

```
this->label4->Text = L"Amount of inclusions:";

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(427, 163);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(121, 17);

this->label5->TabIndex = 10;

this->label5->Text = L"Size of inclusions:";

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(427, 222);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(119, 17);

this->label6->TabIndex = 11;

this->label6->Text = L"Type of inclusion:";

//

// amountOfInclusionsTextBox

//

this->amountOfInclusionsTextBox->Location = System::Drawing::Point(429, 130);

this->amountOfInclusionsTextBox->Name = L"amountOfInclusionsTextBox";

this->amountOfInclusionsTextBox->Size = System::Drawing::Size(140, 22);

this->amountOfInclusionsTextBox->TabIndex = 12;

this->amountOfInclusionsTextBox->Text = L"2";

//

// sizeOfInclusionsTextBox

//

this->sizeOfInclusionsTextBox->Location = System::Drawing::Point(430, 183);

this->sizeOfInclusionsTextBox->Name = L"sizeOfInclusionsTextBox";

this->sizeOfInclusionsTextBox->Size = System::Drawing::Size(139, 22);

this->sizeOfInclusionsTextBox->TabIndex = 13;

this->sizeOfInclusionsTextBox->Text = L"4";
```

```
//

// typeOfInclusionComboBox

//

this->typeOfInclusionComboBox->FormattingEnabled = true;

this->typeOfInclusionComboBox->Items->AddRange(gcnew cli::array< System::Object^  >(2) {
L"circle", L"square" });

this->typeOfInclusionComboBox->Location = System::Drawing::Point(430, 242);

this->typeOfInclusionComboBox->Name = L"typeOfInclusionComboBox";

this->typeOfInclusionComboBox->Size = System::Drawing::Size(139, 24);

this->typeOfInclusionComboBox->TabIndex = 14;

this->typeOfInclusionComboBox->Text = L"circle";

//

// addAfterInclusions

//

this->addAfterInclusions->Enabled = false;

this->addAfterInclusions->Location = System::Drawing::Point(430, 284);

this->addAfterInclusions->Name = L"addAfterInclusions";

this->addAfterInclusions->Size = System::Drawing::Size(138, 45);

this->addAfterInclusions->TabIndex = 15;

this->addAfterInclusions->Text = L"Add after simulation";

this->addAfterInclusions->UseVisualStyleBackColor = true;

this->addAfterInclusions->Click += gcnew System::EventHandler(this, `
&MyForm::addAfterInclusions_Click);

//

// addBeforeInclusions

//

this->addBeforeInclusions->Location = System::Drawing::Point(429, 335);

this->addBeforeInclusions->Name = L"addBeforeInclusions";

this->addBeforeInclusions->Size = System::Drawing::Size(138, 43);

this->addBeforeInclusions->TabIndex = 16;

this->addBeforeInclusions->Text = L"Add before simulation";

this->addBeforeInclusions->UseVisualStyleBackColor = true;

this->addBeforeInclusions->Click += gcnew System::EventHandler(this,
&MyForm::addBeforeInclusions_Click);

//

// neighborhoodMethoodComboBox
```

```
//

this->neighborhoodMethoodComboBox->FormattingEnabled = true;

this->neighborhoodMethoodComboBox->Items->AddRange(gcnew cli::array< System::Object^
>(2) { L"Von Neumann", L"Moore" });

this->neighborhoodMethoodComboBox->Location = System::Drawing::Point(430, 64);

this->neighborhoodMethoodComboBox->Name = L"neighborhoodMethoodComboBox";

this->neighborhoodMethoodComboBox->Size = System::Drawing::Size(139, 24);

this->neighborhoodMethoodComboBox->TabIndex = 17;

this->neighborhoodMethoodComboBox->Text = L"Von Neumann";

//

// label7

//

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(429, 40);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(102, 17);

this->label7->TabIndex = 18;

this->label7->Text = L"Neighborhood:";

//

// label8

//

this->label8->AutoSize = true;

this->label8->Location = System::Drawing::Point(606, 41);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(70, 17);

this->label8->TabIndex = 19;

this->label8->Text = L"Structure:";

//

// structureComboBox

//

this->structureComboBox->FormattingEnabled = true;

this->structureComboBox->Items->AddRange(gcnew cli::array< System::Object^  >(2) {
L"Substructure", L"Dual Phase" });

this->structureComboBox->Location = System::Drawing::Point(609, 64);

this->structureComboBox->Name = L"structureComboBox";

this->structureComboBox->Size = System::Drawing::Size(141, 24);
```

```
this->structureComboBox->TabIndex = 20;

this->structureComboBox->Text = L"Substructure";

//

// structureGenerateButton

//

this->structureGenerateButton->Location = System::Drawing::Point(625, 148);

this->structureGenerateButton->Name = L"structureGenerateButton";

this->structureGenerateButton->Size = System::Drawing::Size(104, 32);

this->structureGenerateButton->TabIndex = 21;

this->structureGenerateButton->Text = L"GENERATE";

this->structureGenerateButton->UseVisualStyleBackColor = true;

this->structureGenerateButton->Click += gcnew System::EventHandler(this,
&MyForm::structureGenerateButton_Click);

//

// label9

//

this->label9->AutoSize = true;

this->label9->Location = System::Drawing::Point(606, 188);

this->label9->Name = L"label9";

this->label9->Size = System::Drawing::Size(111, 17);

this->label9->TabIndex = 22;

this->label9->Text = L"Grains selected:";

//

// grainsSelectedComboBox

//

this->grainsSelectedComboBox->FormattingEnabled = true;

this->grainsSelectedComboBox->Items->AddRange(gcnew cli::array< System::Object^  >(2) {
L"All grains selected", L"N grains selected" });

this->grainsSelectedComboBox->Location = System::Drawing::Point(609, 212);

this->grainsSelectedComboBox->Name = L"grainsSelectedComboBox";

this->grainsSelectedComboBox->Size = System::Drawing::Size(141, 24);

this->grainsSelectedComboBox->TabIndex = 23;

this->grainsSelectedComboBox->Text = L"All grains selected";

this->grainsSelectedComboBox->SelectedIndexChanged += gcnew System::EventHandler(this,
&MyForm::grainsSelectedComboBox_SelectedIndexChanged);

//
```

```
// label10

//

this->label10->AutoSize = true;

this->label10->Location = System::Drawing::Point(609, 239);

this->label10->Name = L"label10";

this->label10->Size = System::Drawing::Size(61, 17);

this->label10->TabIndex = 24;

this->label10->Text = L"GB size:";

//

// gbSizeTextBox

//

this->gbSizeTextBox->Location = System::Drawing::Point(609, 259);

this->gbSizeTextBox->Name = L"gbSizeTextBox";

this->gbSizeTextBox->Size = System::Drawing::Size(141, 22);

this->gbSizeTextBox->TabIndex = 25;

this->gbSizeTextBox->Text = L"1";

//

// label11

//

this->label11->AutoSize = true;

this->label11->Location = System::Drawing::Point(609, 284);

this->label11->Name = L"label11";

this->label11->Size = System::Drawing::Size(83, 17);

this->label11->TabIndex = 26;

this->label11->Text = L"GB amount:";

//

// gbAmountTextBox

//

this->gbAmountTextBox->Location = System::Drawing::Point(609, 307);

this->gbAmountTextBox->Name = L"gbAmountTextBox";

this->gbAmountTextBox->Size = System::Drawing::Size(141, 22);

this->gbAmountTextBox->TabIndex = 27;

this->gbAmountTextBox->Text = L"1";

this->gbAmountTextBox->TextChanged += gcnew System::EventHandler(this,
&MyForm::gbAmountTextBox_TextChanged);

//
```

```
// generateGBButton

//

this->generateGBButton->Enabled = false;

this->generateGBButton->Location = System::Drawing::Point(625, 335);

this->generateGBButton->Name = L"generateGBButton";

this->generateGBButton->Size = System::Drawing::Size(104, 33);

this->generateGBButton->TabIndex = 28;

this->generateGBButton->Text = L"Generate GB";

this->generateGBButton->UseVisualStyleBackColor = true;

this->generateGBButton->Click += gcnew System::EventHandler(this,
&MyForm::generateGBButton_Click);

//

// clearSpaceButton

//

this->clearSpaceButton->Location = System::Drawing::Point(625, 375);

this->clearSpaceButton->Name = L"clearSpaceButton";

this->clearSpaceButton->Size = System::Drawing::Size(104, 37);

this->clearSpaceButton->TabIndex = 29;

this->clearSpaceButton->Text = L"Clear space";

this->clearSpaceButton->UseVisualStyleBackColor = true;

this->clearSpaceButton->Click += gcnew System::EventHandler(this,
&MyForm::clearSpaceButton_Click);

//

// clearAll

//

this->clearAll->BackColor = System::Drawing::SystemColors::AppWorkspace;

this->clearAll->Enabled = false;

this->clearAll->Location = System::Drawing::Point(609, 418);

this->clearAll->Name = L"clearAll";

this->clearAll->Size = System::Drawing::Size(141, 60);

this->clearAll->TabIndex = 30;

this->clearAll->Text = L"CLEAR ALL";

this->clearAll->UseVisualStyleBackColor = false;

this->clearAll->Click += gcnew System::EventHandler(this, &MyForm::clearAll_Click);

//

// label12
```

```
//

this->label12->AutoSize = true;

this->label12->Location = System::Drawing::Point(609, 95);

this->label12->Name = L"label12";

this->label12->Size = System::Drawing::Size(96, 17);

this->label12->TabIndex = 31;

this->label12->Text = L"Amount grain:";

//

// amountGrainDomainUpDown

//

this->amountGrainDomainUpDown->Items->Add(L"20");

this->amountGrainDomainUpDown->Items->Add(L"19");

this->amountGrainDomainUpDown->Items->Add(L"18");

this->amountGrainDomainUpDown->Items->Add(L"17");

this->amountGrainDomainUpDown->Items->Add(L"16");

this->amountGrainDomainUpDown->Items->Add(L"15");

this->amountGrainDomainUpDown->Items->Add(L"14");

this->amountGrainDomainUpDown->Items->Add(L"13");

this->amountGrainDomainUpDown->Items->Add(L"12");

this->amountGrainDomainUpDown->Items->Add(L"11");

this->amountGrainDomainUpDown->Items->Add(L"10");

this->amountGrainDomainUpDown->Items->Add(L"9");

this->amountGrainDomainUpDown->Items->Add(L"8");

this->amountGrainDomainUpDown->Items->Add(L"7");

this->amountGrainDomainUpDown->Items->Add(L"6");

this->amountGrainDomainUpDown->Items->Add(L"5");

this->amountGrainDomainUpDown->Items->Add(L"4");

this->amountGrainDomainUpDown->Items->Add(L"3");

this->amountGrainDomainUpDown->Items->Add(L"2");

this->amountGrainDomainUpDown->Items->Add(L"1");

this->amountGrainDomainUpDown->Location = System::Drawing::Point(609, 116);

this->amountGrainDomainUpDown->Name = L"amountGrainDomainUpDown";

this->amountGrainDomainUpDown->Size = System::Drawing::Size(141, 22);

this->amountGrainDomainUpDown->TabIndex = 32;

this->amountGrainDomainUpDown->Text = L"1";
```

```
//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::SystemColors::ActiveBorder;

this->ClientSize = System::Drawing::Size(775, 484);

this->Controls->Add(this->amountGrainDomainUpDown);

this->Controls->Add(this->label12);

this->Controls->Add(this->clearAll);

this->Controls->Add(this->clearSpaceButton);

this->Controls->Add(this->generateGBButton);

this->Controls->Add(this->gbAmountTextBox);

this->Controls->Add(this->label11);

this->Controls->Add(this->gbSizeTextBox);

this->Controls->Add(this->label10);

this->Controls->Add(this->grainsSelectedComboBox);

this->Controls->Add(this->label9);

this->Controls->Add(this->structureGenerateButton);

this->Controls->Add(this->structureComboBox);

this->Controls->Add(this->label8);

this->Controls->Add(this->label7);

this->Controls->Add(this->neighborhoodMethoodComboBox);

this->Controls->Add(this->addBeforeInclusions);

this->Controls->Add(this->addAfterInclusions);

this->Controls->Add(this->panel1);

this->Controls->Add(this->typeOfInclusionComboBox);

this->Controls->Add(this->sizeOfInclusionsTextBox);

this->Controls->Add(this->amountOfInclusionsTextBox);

this->Controls->Add(this->label6);

this->Controls->Add(this->label5);

this->Controls->Add(this->label4);

this->Controls->Add(this->amoutNucleonDomianUpDown);

this->Controls->Add(this->ySizeValueTextBox);

this->Controls->Add(this->xSizeValueTextBox);
```

```
        this->Controls->Add(this->startSymulationButton);
        this->Controls->Add(this->label3);
        this->Controls->Add(this->label2);
        this->Controls->Add(this->label1);
        this->Controls->Add(this->menuStrip1);
        this->Cursor = System::Windows::Forms::Cursors::Arrow;
        this->MainMenuStrip = this->menuStrip1;
        this->MaximizeBox = false;
        this->Name = L"MyForm";
        this->Text = L"Simulation Multiscale Modelling";
        this->menuStrip1->ResumeLayout(false);
        this->menuStrip1->PerformLayout();
        this->panel1->ResumeLayout(false);
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->EndInit();
        this->ResumeLayout(false);
        this->PerformLayout();

}
#pragma endregion
public:
int rowTable = 0;
int colTable = 0;
int **table;
int **tmpTable;
int ** tableGB;
int *tablePhase;
int nucleonRow = 0;
int nucleonCol = 0;
int amountNucleon = 0;
int amountInclusions = 0;
int sizeInclusions = 0;
int idNucleon = 0;
int gbSize = 0;
String ^ xSizeValueFromTextBox = "";
String ^ ySizeValueFromTextBox = "";
String ^ amoutNucleonFromDomianUpDown = "";

String ^ amountOfInclusionsFromTextBox = "";

String ^ sizeOfInclusionsFromTextBox = "";

String ^ typeOfInclusionFromComboBox = "";

String ^ neighborhoodMethoodFromComboBox = "";

String ^ grainsSelectedFromComboBox = "";

String ^ gbSizeFromTextBox = "";

String ^ gbAmountFromTextBox = "";

String ^ structureFromComboBox = "";

String ^ amountGrainFromDomainUpDown = "";


DomainUpDown ^ domainUpDown;

Graphics ^ board;

Bitmap ^ image1;

bool tableIsExist = false;


void createTable(int row, int col)

{

        // Table 1

        table = new int *[row];

        for (int i = 0; i < row; i++)

        {

                table[i] = new int[col];

        }

        for (int i = 0; i < row; i++)

        {

                for (int j = 0; j < col; j++)

                {

                        table[i][j] = 0;

                }

        }

        // Table 2

        tmpTable = new int *[row];

        for (int i = 0; i < row; i++)

        {
                tmpTable[i] = new int[col];
        }
        for (int i = 0; i < row; i++)
        {
                for (int j = 0; j < col; j++)
                {
                        tmpTable[i][j] = 0;
                }
        }
        // Table GB
        tableGB = new int *[row];
        for (int i = 0; i < row; i++)
        {
                tableGB[i] = new int[col];
        }
        for (int i = 0; i < row; i++)
        {
                for (int j = 0; j < col; j++)
                {
                        tableGB[i][j] = 0;
                }
        }

        tableIsExist = true;
}
void deleteTable(int row, int col)
{
        for (int i = 0; i < row; i++)
        {
                delete[] table[i];
        }
        delete[] table;

        for (int i = 0; i < row; i++)
        {
                delete[] tmpTable[i];
        }
        delete[] tmpTable;

        for (int i = 0; i < row; i++)
        {
                delete[] tableGB[i];
        }
        delete[] tableGB;

        tableIsExist = false;
}

int randPositionXNucleon()
{
        return rand() % colTable;
}

int randPositionYNucleon()
{
        return rand() % rowTable;
}

void drawNucleon(int x, int y, int colorPencil)
{
        Pen ^ pencil = gcnew Pen(System::Drawing::Color::Navy);
        pencil->Width = 1;
        if (colorPencil == 1)
        {
                pencil->Color = System::Drawing::Color::Red;
        }
        else if (colorPencil == 2)
        {
                pencil->Color = System::Drawing::Color::Green;
        }
```

```
else if (colorPencil == 3)
{
        pencil->Color = System::Drawing::Color::Blue;
}
else if (colorPencil == 4)
{
        pencil->Color = System::Drawing::Color::Brown;
}
else if (colorPencil == 5)
{
        pencil->Color = System::Drawing::Color::Bisque;
}
else if (colorPencil == 6)
{
        pencil->Color = System::Drawing::Color::BlanchedAlmond;
}
else if (colorPencil == 7)
{
        pencil->Color = System::Drawing::Color::Chocolate;
}
else if (colorPencil == 8)
{
        pencil->Color = System::Drawing::Color::Cornsilk;
}
else if (colorPencil == 9)
{
        pencil->Color = System::Drawing::Color::DarkKhaki;
}
else if (colorPencil == 10)
{
        pencil->Color = System::Drawing::Color::DarkOrange;
}
else if (colorPencil == 11)
{
        pencil->Color = System::Drawing::Color::DarkOliveGreen;
```

```
}
else if (colorPencil == 12)
{
	pencil->Color = System::Drawing::Color::AliceBlue;
}
else if (colorPencil == 13)
{
	pencil->Color = System::Drawing::Color::AntiqueWhite;
}
else if (colorPencil == 14)
{
	pencil->Color = System::Drawing::Color::Aqua;
}
else if (colorPencil == 15)
{
	pencil->Color = System::Drawing::Color::BlueViolet;
}
else if (colorPencil == 16)
{
	pencil->Color = System::Drawing::Color::Cornsilk;
}
else if (colorPencil == 17)
{
	pencil->Color = System::Drawing::Color::DarkSalmon;
}
else if (colorPencil == 18)
{
	pencil->Color = System::Drawing::Color::ForestGreen;
}
else if (colorPencil == 19)
{
	pencil->Color = System::Drawing::Color::Goldenrod;
}
else if (colorPencil == 20)
{
```

```
            pencil->Color = System::Drawing::Color::GreenYellow;
        }
        else if (colorPencil == 100)
        {
            pencil->Color = System::Drawing::Color::Black;
        }



        board->DrawRectangle(pencil, y, x, 1, 1);
        Graphics ^ picture = Graphics::FromImage(image1);
        picture->DrawRectangle(pencil, y, x, 1, 1);
    }
    void checkNeighbors(int actualPositionX, int actualPositionY, int id)
    {
        if (actualPositionX - 1 >= 0)
        {
            if (tmpTable[actualPositionX - 1][actualPositionY] == 0)
            {
                tmpTable[actualPositionX - 1][actualPositionY] = id;
            }
        }
        if (actualPositionX + 1 < rowTable)
        {
            if (tmpTable[actualPositionX + 1][actualPositionY] == 0)
            {
                tmpTable[actualPositionX + 1][actualPositionY] = id;
            }
        }
        if (actualPositionY - 1 >= 0)
        {
            if (tmpTable[actualPositionX][actualPositionY - 1] == 0)
            {
                tmpTable[actualPositionX][actualPositionY - 1] = id;
            }
        }
```

```
		}

		if (actualPositionY + 1 < colTable)
		{
			if (tmpTable[actualPositionX][actualPositionY + 1] == 0)
			{
				tmpTable[actualPositionX][actualPositionY + 1] = id;
			}
		}
	}
	// Von Neumann neighborhood Methood
	void checkNeighborsVonNeumann(int actualPositionX, int actualPositionY, int id)
	{
		int counter = 0;
		if (actualPositionX - 1 >= 0)
		{
			if (tmpTable[actualPositionX - 1][actualPositionY] == id)
			{
				counter++;
			}
		}
		if (actualPositionX + 1 < rowTable)
		{
			if (tmpTable[actualPositionX + 1][actualPositionY] == id)
			{
				counter++;
			}
		}
		if (actualPositionY - 1 >= 0)
		{
			if (tmpTable[actualPositionX][actualPositionY - 1] == id)
			{
				counter++;
			}
		}
		if (actualPositionY + 1 < colTable)
```

```
                {
                        if (tmpTable[actualPositionX][actualPositionY + 1] == id)
                        {
                                counter++;
                        }
                }
                if (counter == 4)
                {
                        tmpTable[actualPositionX][actualPositionY] == id;
                }
        }
        // Moore neighborhood Methood
        void checkNeighborsMoore(int actualPositionX, int actualPositionY, int id)
        {
                if (actualPositionX > rowTable - 3)
                {
                        actualPositionX = actualPositionX - 2;
                }

                if (actualPositionY > colTable - 3)
                {
                        actualPositionY = actualPositionY - 2;
                }
if (tmpTable[actualPositionX][actualPositionY] == id && tmpTable[actualPositionX + 1][actualPositionY] == id &&
tmpTable[actualPositionX + 2][actualPositionY] == id
                        && tmpTable[actualPositionX + 2][actualPositionY + 1] == id)
                {
                        tmpTable[actualPositionX + 1][actualPositionY + 1] = id;
                }

                if (tmpTable[actualPositionX][actualPositionY] == id && tmpTable[actualPositionX +
                1][actualPositionY + 1] == id && tmpTable[actualPositionX + 2][actualPositionY] == id)
                {
                        tmpTable[actualPositionX + 1][actualPositionY] = id;
                }
        if (tmpTable[actualPositionX][actualPositionY] == id &&
        tmpTable[actualPositionX][actualPositionY + 2] == id && tmpTable[actualPositionX +
        2][actualPositionY] == id)

        {

                tmpTable[actualPositionX + 1][actualPositionY + 1] == id;

        }


        int probability = 10;

        int tableColor[8] = { 0 };


        tableColor[0] = tmpTable[actualPositionX][actualPositionY];

        tableColor[1] = tmpTable[actualPositionX + 1][actualPositionY];

        tableColor[2] = tmpTable[actualPositionX + 2][actualPositionY];

        tableColor[3] = tmpTable[actualPositionX][actualPositionY + 1];

        tableColor[4] = tmpTable[actualPositionX][actualPositionY + 2];

        tableColor[5] = tmpTable[actualPositionX + 1][actualPositionY + 1];

        tableColor[6] = tmpTable[actualPositionX + 2][actualPositionY + 2];

        tableColor[7] = tmpTable[actualPositionX + 2][actualPositionY + 1];


        int mostPopularColor = rand() % 8 + 1;


        if (rand() % 100 + 1 <= 90)

        {

                tmpTable[actualPositionX + 1][actualPositionY + 1] == mostPopularColor;

        }


}

void growNucleons()

{

        for (int i = 0; i < rowTable; i++)

        {

                for (int j = 0; j < colTable; j++)

                {

                        tmpTable[i][j] = table[i][j];

                }

        }
        for (int amount = 1; amount <= amountNucleon; amount++)
        {
                for (int i = 0; i < rowTable; i++)
                {
                        for (int j = 0; j < colTable; j++)
                        {
                                if (table[i][j] == amount)
                                {
                                        if (neighborhoodMethoodFromComboBox == "Von
                                        Neumann")
                                        {
                                                checkNeighbors(i, j, amount);
                                                checkNeighborsVonNeumann(i, j, amount);
                                        }                       else if
                                                (neighborhoodMethoodFromComboBox ==
                                                "Moore")
                                        {
                                                checkNeighbors(i, j, amount);
                                                checkNeighborsMoore(i, j, amount);
                                        }
                                }
                        }
                }
        }

        for (int i = 0; i < rowTable; i++)
        {
                for (int j = 0; j < colTable; j++)
                {
                        table[i][j] = tmpTable[i][j];
                }
        }
}
void saveMicrostrure()
{
        Stream^ fileName;
        SaveFileDialog^ saveFileDialog1 = gcnew SaveFileDialog;
        saveFileDialog1->Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
        saveFileDialog1->FilterIndex = 2;
        saveFileDialog1->RestoreDirectory = true;
        if (saveFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
        {
                if ((fileName = saveFileDialog1->OpenFile()) != nullptr)
                {
                        StreamWriter^ saveFile = gcnew StreamWriter(fileName);
                        String ^ r;
                        String ^ c;
                        String ^ phase = Convert::ToString(0);
                        String ^ id;
                        String ^ allText = "";
                        for (int i = 0; i < rowTable; i++)
                        {
                                for (int j = 0; j < colTable; j++)
                                {
                                        r = Convert::ToString(i);
                                        c = Convert::ToString(j);
                                        id = Convert::ToString(table[i][j]);
                                        allText = r + " " + c + " " + phase + " " + id;
                                        saveFile->WriteLine(allText);
                                }
                        }
                        saveFile->Flush();
                        saveFile->Close();
                }
        }
}
void saveMicrostrureBMP()
{
        SaveFileDialog^ saveDiag2 = gcnew SaveFileDialog();
        saveDiag2->Filter = "Dateityp BMP (*.bmp)|*.bmp|All files (*.*)|*.*";
        saveDiag2->FilterIndex = 1;
```

```
          saveDiag2->RestoreDirectory = true;
          pictureBox1->Image = image1;
          if (saveDiag2->ShowDialog() == System::Windows::Forms::DialogResult::OK)
          {
                    String^ savePath = saveDiag2->FileName;
                              pictureBox1->Image->Save(savePath);
          }
}
void loadFile()
{
          Stream^ myStream;
          OpenFileDialog^ openFileDialog1 = gcnew OpenFileDialog;

          openFileDialog1->InitialDirectory = "c:\\";
          openFileDialog1->Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
          openFileDialog1->FilterIndex = 2;
          openFileDialog1->RestoreDirectory = true;

          if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
          {
                    if ((myStream = openFileDialog1->OpenFile()) != nullptr)
                    {
                              String ^ fileName;
                              fileName = openFileDialog1->FileName;
                              StreamReader^ din = File::OpenText(fileName);
                              String ^ str;
                              String^ delimStr = " ,.:\t";
                              array<Char>^ delimiter = delimStr->ToCharArray();
                              int sizeTable = 300;
                              createTable(sizeTable, sizeTable);
                              array<String^>^ words;
                              while ((str = din->ReadLine()) != nullptr)
                              {
                                        words = str->Split(delimiter);
                                        for (int word = 0; word < words->Length; word++)
```

```
{

        if (!words[word]->Length) // skip empty words
                continue;
}
int index = 0;
int row = Convert::ToInt32(words[index]);
int col = Convert::ToInt32(words[index + 1]);
int phase = Convert::ToInt32(words[index + 2]);
int valueNucleon = Convert::ToInt32(words[index + 3]);
table[row][col] = valueNucleon;
pictureBox1->Image = image1;
Pen ^ pencil = gcnew Pen(System::Drawing::Color::Navy);
pencil->Width = 1;
if (table[row][col] == 1)
{
        pencil->Color = System::Drawing::Color::Red;
}
else if (table[row][col] == 2)
{
        pencil->Color = System::Drawing::Color::Green;
}
else if (table[row][col] == 3)
{
        pencil->Color = System::Drawing::Color::Blue;
}
else if (table[row][col] == 4)
{
        pencil->Color = System::Drawing::Color::Brown;
}
else if (table[row][col] == 5)
{
        pencil->Color = System::Drawing::Color::Bisque;
}
else if (table[row][col] == 6)
```

```
{
        pencil->Color =
System::Drawing::Color::BlanchedAlmond;
}
else if (table[row][col] == 7)
{
        pencil->Color = System::Drawing::Color::Chocolate;
}
else if (table[row][col] == 8)
{
        pencil->Color = System::Drawing::Color::Cornsilk;
}
else if (table[row][col] == 9)
{
        pencil->Color = System::Drawing::Color::DarkKhaki;
}
else if (table[row][col] == 10)
{
        pencil->Color = System::Drawing::Color::DarkOrange;
}
else if (table[row][col] == 11)
{
        pencil->Color =
System::Drawing::Color::DarkOliveGreen;
}
else if (table[row][col] == 12)
{
        pencil->Color = System::Drawing::Color::AliceBlue;
}
else if (table[row][col] == 13)
{
        pencil->Color = System::Drawing::Color::AntiqueWhite;
}
else if (table[row][col] == 14)
{
        pencil->Color = System::Drawing::Color::Aqua;
```

```
                                    }
                                    else if (table[row][col] == 15)
                                    {
                                        pencil->Color = System::Drawing::Color::BlueViolet;
                                    }
                                    else if (table[row][col] == 16)
                                    {
                                        pencil->Color = System::Drawing::Color::Cornsilk;
                                    }
                                    else if (table[row][col] == 17)
                                    {
                                        pencil->Color = System::Drawing::Color::DarkSalmon;
                                    }
                                    else if (table[row][col] == 18)
                                    {
                                        pencil->Color = System::Drawing::Color::ForestGreen;
                                    }
                                    else if (table[row][col] == 19)
                                    {
                                        pencil->Color = System::Drawing::Color::Goldenrod;
                                    }
                                    else if (table[row][col] == 20)
                                    {
                                        pencil->Color = System::Drawing::Color::GreenYellow;
                                    }
                                    else if (table[row][col] == 100)
                                    {
                                        pencil->Color = System::Drawing::Color::Black;
                                    }
                                    board = panel1->CreateGraphics();
                                    board->DrawRectangle(pencil, col, row, 1, 1);
                                }
                                myStream->Close();
                            }
                        }
```

```
}
        void loadFileBitmap()
        {
                Stream^ myStream;
                OpenFileDialog^ openFileDialog1 = gcnew OpenFileDialog;
                openFileDialog1->InitialDirectory = "c:\\";
                openFileDialog1->Filter = "bmp files (*.bmp)|*.bmp|All files (*.*)|*.*";
                openFileDialog1->FilterIndex = 2;
                openFileDialog1->RestoreDirectory = true;

                if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
                {
                        if ((myStream = openFileDialog1->OpenFile()) != nullptr)
                        {
                                String ^ fileName;
                                fileName = openFileDialog1->FileName;
                                panel1->BackgroundImage = Image::FromFile(fileName);
                                Bitmap^ myBitmap = gcnew Bitmap(fileName);
                                createTable(myBitmap->Width, myBitmap->Height);
                                for (int i = 0; i < myBitmap->Width; i++)
                                {
                                        for (int j = 0; j < myBitmap->Height; j++)
                                        {
                                                Color colorImage = myBitmap->GetPixel(i, j);
                                        if (System::Drawing::Color::Red == colorImage)
                                                {
                                                        table[i][j] = 1;
                                                }
                                        else if (System::Drawing::Color::Green == colorImage)
                                                {
                                                        table[i][j] = 2;
                                                }
                                        else if (System::Drawing::Color::Blue == colorImage)
                                                {
                                                        table[i][j] = 3;
```

```
                    }
                else if (System::Drawing::Color::Brown == colorImage)
                    {
                        table[i][j] = 4;
                    }
                else if (System::Drawing::Color::Bisque == colorImage)
                    {
                        table[i][j] = 5;
                    }
            else if (System::Drawing::Color::BlanchedAlmond == colorImage)
                    {
                        table[i][j] = 6;
                    }
            else if (System::Drawing::Color::Chocolate == colorImage)
                    {
                        table[i][j] = 7;
                    }
            else if (System::Drawing::Color::Cornsilk == colorImage)
                    {
                        table[i][j] = 8;
                    }
            else if (System::Drawing::Color::DarkKhaki == colorImage)
                    {
                        table[i][j] = 9;
                    }
            else if (System::Drawing::Color::DarkOrange == colorImage)
                    {
                        table[i][j] = 10;
                    }
            else if (System::Drawing::Color::DarkOliveGreen == colorImage)
                    {
                        table[i][j] = 11;
                    }
            else if (System::Drawing::Color::AliceBlue == colorImage)
                    {
```

```
            table[i][j] = 12;
        }
else if (System::Drawing::Color::AntiqueWhite == colorImage)
        {
            table[i][j] = 13;
        }
else if (System::Drawing::Color::Aqua == colorImage)
        {
            table[i][j] = 14;
        }
else if (System::Drawing::Color::BlueViolet == colorImage)
        {
            table[i][j] = 15;
        }
else if (System::Drawing::Color::Cornsilk == colorImage)
        {
            table[i][j] = 16;
        }
else if (System::Drawing::Color::DarkSalmon == colorImage)
        {
            table[i][j] = 17;
        }
else if (System::Drawing::Color::ForestGreen == colorImage)
        {
            table[i][j] = 18;
        }
else if (System::Drawing::Color::Goldenrod == colorImage)
        {
            table[i][j] = 19;
        }
else if (System::Drawing::Color::GreenYellow == colorImage)
        {
            table[i][j] = 20;
        }
    }
```

```
                                                }
                                        }
                                }
                        }

                void drawInclusions(int sizeDraw, String ^ option, int x, int y)
                {
                        Pen ^ pencil = gcnew Pen(System::Drawing::Color::Navy);
                        pencil->Width = sizeDraw;
                        pencil->Color = System::Drawing::Color::Black;
                        Graphics ^ picture = Graphics::FromImage(image1);
                        if (option == "circle")
                        {
                                board->DrawEllipse(pencil, y, x, sizeDraw, sizeDraw);
                                picture->DrawEllipse(pencil, y, x, sizeDraw, sizeDraw);
                        }
                        else if (option == "square")
                        {
                                board->DrawRectangle(pencil, y, x, sizeDraw, sizeDraw);
                                picture->DrawRectangle(pencil, y, x, sizeDraw, sizeDraw);
                        }
                }

        private: System::Void startSymulationButton_Click(System::Object^  sender, System::EventArgs^  e)
        {
                srand(time(NULL));
                addAfterInclusions->Enabled = true;

                bool testxSizeValueFromTextBox = true;
                bool testySizeValueFromTextBox = true;
                bool testamoutNucleonDomianUpDown = true;
                bool testamountOfInclusionsFromTextBox = true;
                bool testsizeOfInclusionsFromTextBox = true;
                bool testtypeOfInclusionFromComboBox = true;
                bool testNeighborhoodMethoodFromComboBox = true;
```

```
xSizeValueFromTextBox = xSizeValueTextBox->Text;

if (xSizeValueFromTextBox == "" || xSizeValueFromTextBox == "0")
{
        MessageBox::Show("Options x Size Error. Please set!");
        testxSizeValueFromTextBox = false;
}

ySizeValueFromTextBox = ySizeValueTextBox->Text;

if (ySizeValueFromTextBox == "" || ySizeValueFromTextBox == "0")
{
        MessageBox::Show("Options y Size Error. Please set!");
        testySizeValueFromTextBox = false;
}

amoutNucleonFromDomianUpDown = amoutNucleonDomianUpDown->Text;

if (amoutNucleonFromDomianUpDown == "" || amoutNucleonFromDomianUpDown == "0")
{
        MessageBox::Show("Options Nucleon Amout Error. Please set!");
        testamoutNucleonDomianUpDown = false;
}

amountOfInclusionsFromTextBox = amountOfInclusionsTextBox->Text;

if (amountOfInclusionsFromTextBox == "")
{
        MessageBox::Show("Options Amount of Inclusions Error. Please set!");
        testamountOfInclusionsFromTextBox = false;
}

sizeOfInclusionsFromTextBox = sizeOfInclusionsTextBox->Text;
```

```
if (sizeOfInclusionsFromTextBox == "")

{

        MessageBox::Show("Options Size of Inclusions Error. Please set!");

        testsizeOfInclusionsFromTextBox = false;

}


typeOfInclusionFromComboBox = typeOfInclusionComboBox->Text;


if (typeOfInclusionFromComboBox == "")

{

        MessageBox::Show("Options Type of Inclusion Error. Please set!");

        testtypeOfInclusionFromComboBox = false;

}


neighborhoodMethoodFromComboBox = neighborhoodMethoodComboBox->Text;


if (neighborhoodMethoodFromComboBox == "")

{

        MessageBox::Show("Options Type of Neighborhood Methood Error. Please set!");

        testNeighborhoodMethoodFromComboBox = false;

}




if (testxSizeValueFromTextBox == true && testySizeValueFromTextBox == true &&
testamoutNucleonDomianUpDown == true && testamountOfInclusionsFromTextBox == true &&
testsizeOfInclusionsFromTextBox == true && testtypeOfInclusionFromComboBox == true &&
testNeighborhoodMethoodFromComboBox == true)

{

        rowTable = Convert::ToInt32(xSizeValueFromTextBox);

        colTable = Convert::ToInt32(ySizeValueFromTextBox);

        amountNucleon = Convert::ToInt32(amoutNucleonFromDomianUpDown);


        if (tableIsExist == false)

        {

                createTable(rowTable, colTable);

        }
```

```
board = panel1->CreateGraphics();
image1 = gcnew Bitmap(panel1->Width, panel1->Height);

// Randome value index table and inicjalization nucleon on this table
int id = rand() % amountNucleon + 1;
tablePhase = new int[amountNucleon + 1];

for (int i = 0; i < amountNucleon + 1; i++)
{
		tablePhase[i] = 0;
}

for (int i = 0; i < amountNucleon; i++)
{
		nucleonCol = randPositionXNucleon();
		nucleonRow = randPositionYNucleon();
		table[nucleonRow][nucleonCol] = id;
		tablePhase[id] = tablePhase[id] + 1;
		id = rand() % amountNucleon + 1;
}

for (int i = 0; i < 1000; i++)
{
		growNucleons();
}

// Draw nucleon
for (int amount = 1; amount <= amountNucleon; amount++)
{
		for (int i = 0; i < rowTable; i++)
		{
				for (int j = 0; j < colTable; j++)
				{
                    }

                        if (table[i][j] == amount)
                        {
                            drawNucleon(j, i, amount);
                        }
                    }
                }
            }

            for (int i = 0; i < rowTable; i++)
            {
                for (int j = 0; j < colTable; j++)
                {
                    if (table[i][j] == 100)
                    {
        drawInclusions(Convert::ToInt32(sizeOfInclusionsFromTextBox), typeOfInclusionFromComboBox, j, i);
                    }
                }
            }

            addBeforeInclusions->Enabled = false;

            if (tableIsExist == true)
            {
                clearAll->Enabled = true;
                generateGBButton->Enabled = true;
            }
            else
            {
                clearAll->Enabled = false;
                generateGBButton->Enabled = false;
            }
        }
    }

private: System::Void toTXTToolStripMenuItem_Click(System::Object^  sender, System::EventArgs^  e)
```

```
{
				saveMicrostrure();
				for (int i = 0; i < rowTable; i++)
				{
					delete [] table[i];
				}
				delete [] table;

				for (int i = 0; i < rowTable; i++)
				{
					delete [] tmpTable[i];
				}
				delete [] tmpTable;

				delete [] tablePhase;
}

private: System::Void toBMPToolStripMenuItem_Click(System::Object^  sender, System::EventArgs^  e)
{
				saveMicrostrureBMP();
				for (int i = 0; i < rowTable; i++)
				{
					delete[] table[i];
				}
				delete[] table;
}

private: System::Void FromTXTToolStripMenuItem_Click(System::Object^  sender, System::EventArgs^  e)
{
				loadFile();
}

private: System::Void fromBitmapToolStripMenuItem_Click(System::Object^  sender, System::EventArgs^  e)
{
				loadFileBitmap();
```

```
}


private: System::Void addBeforeInclusions_Click(System::Object^  sender, System::EventArgs^  e)
{
                        amountOfInclusionsFromTextBox = amountOfInclusionsTextBox->Text;
                        amountInclusions = Convert::ToInt32(amountOfInclusionsFromTextBox);

                        sizeOfInclusionsFromTextBox = sizeOfInclusionsTextBox->Text;
                        sizeInclusions = Convert::ToInt32(sizeOfInclusionsFromTextBox);

                        xSizeValueFromTextBox = xSizeValueTextBox->Text;
                        rowTable = Convert::ToInt32(xSizeValueFromTextBox);

                        ySizeValueFromTextBox = ySizeValueTextBox->Text;
                        colTable = Convert::ToInt32(ySizeValueFromTextBox);

                        typeOfInclusionFromComboBox = typeOfInclusionComboBox->Text;
                        if (tableIsExist == false)
                        {
                                createTable(rowTable, colTable);
                        }
                        board = panel1->CreateGraphics();
                        image1 = gcnew Bitmap(panel1->Width, panel1->Height);

                        for (int i = 1; i <= amountInclusions; i++)
                        {
                                int inclusionsCol = randPositionXNucleon();
                                int InclusionsRow = randPositionYNucleon();
                                table[InclusionsRow][inclusionsCol] = 100;
                        drawInclusions(sizeInclusions, typeOfInclusionFromComboBox, inclusionsCol, InclusionsRow);
                        }
}
        private: System::Void addAfterInclusions_Click(System::Object^  sender, System::EventArgs^  e)
{
                        amountOfInclusionsFromTextBox = amountOfInclusionsTextBox->Text;
```

```
amountInclusions = Convert::ToInt32(amountOfInclusionsFromTextBox);


sizeOfInclusionsFromTextBox = sizeOfInclusionsTextBox->Text;
sizeInclusions = Convert::ToInt32(sizeOfInclusionsFromTextBox);


xSizeValueFromTextBox = xSizeValueTextBox->Text;
rowTable = Convert::ToInt32(xSizeValueFromTextBox);


ySizeValueFromTextBox = ySizeValueTextBox->Text;
colTable = Convert::ToInt32(ySizeValueFromTextBox);


typeOfInclusionFromComboBox = typeOfInclusionComboBox->Text;
if (tableIsExist == false)
{
        createTable(rowTable, colTable);
}


board = panel1->CreateGraphics();
image1 = gcnew Bitmap(panel1->Width, panel1->Height);


int inclusionsCol;
int inclusionsRow;


for (int i = 1; i <= amountInclusions; i++)
{
        do
        {


                inclusionsCol = randPositionXNucleon() - 2;
                inclusionsRow = randPositionYNucleon();


                while (inclusionsCol < 1 )
                {
                        inclusionsCol = randPositionXNucleon();
                }
```

```
                                        while (inclusionsCol == colTable - 1)
                                        {
                                                inclusionsCol = randPositionXNucleon();
                                        }

                                        while (inclusionsRow < 1)
                                        {
                                                inclusionsRow = randPositionYNucleon();
                                        }

                                        while (inclusionsRow == rowTable - 1)
                                        {
                                                inclusionsRow = randPositionYNucleon();
                                        }
                                } while (table[inclusionsRow][inclusionsCol] == table[inclusionsRow][inclusionsCol + 1]);

                                        table[inclusionsRow][inclusionsCol] = 100;
                        drawInclusions(sizeInclusions, typeOfInclusionFromComboBox, inclusionsCol, inclusionsRow);
                                }
}
private: System::Void clearAll_Click(System::Object^  sender, System::EventArgs^  e)
{
                                deleteTable(rowTable, colTable);
                                board->Clear(SystemColors::ControlLight);
                                clearAll->Enabled = false;
}
private: System::Void generateGBButton_Click(System::Object^  sender, System::EventArgs^  e)
{
                                bool testgrainsSelectedFromComboBox = true;
                                bool testgbSizeTextBox = true;
                                bool testgbAmountTextBox = true;
                                int id = 0;
                                int amount = 0;
                                grainsSelectedFromComboBox = grainsSelectedComboBox->Text;
```

```
if (grainsSelectedFromComboBox == "")
{
	MessageBox::Show("Options Grains selected Error. Please set!");
	testgrainsSelectedFromComboBox = false;
}

gbSizeFromTextBox = gbSizeTextBox->Text;

if (gbSizeFromTextBox == "")
{
	MessageBox::Show("Options GB size Error. Please set!");
	testgbSizeTextBox = false;
}

Pen ^ pencil = gcnew Pen(System::Drawing::Color::Navy);
pencil->Color = System::Drawing::Color::Black;
Graphics ^ picture = Graphics::FromImage(image1);

if (testgrainsSelectedFromComboBox == true && testgbSizeTextBox == true)
{
	gbSize = Convert::ToInt32(gbSizeFromTextBox);
	pencil->Width = gbSize;
	if (grainsSelectedFromComboBox == "All grains selected")
	{
		for (int i = 0; i < rowTable - 1; i++)
		{
			for (int j = 0; j < colTable - 1; j++)
			{
				if (table[i][j] != table[i][j + 1])
				{
					table[i][j] = 100;
					tableGB[i][j] = table[i][j];
				board->DrawRectangle(pencil, i, j, gbSize, gbSize);
				}
```

```
                                        if (table[i][j] != table[i + 1][j])
                                        {
                                                table[i][j] = 100;
                                                tableGB[i][j] = table[i][j];
                                        board->DrawRectangle(pencil, i, j, gbSize, gbSize);
                                        }
                                }
                        }

                }
                else if (grainsSelectedFromComboBox == "N grains selected")
                {
                        gbAmountFromTextBox = gbAmountTextBox->Text;
                        if (gbAmountFromTextBox == " ")
                        {
                                MessageBox::Show("Options GB amount Error. Please set!");
                                testgbAmountTextBox = false;
                        }

if (testgbAmountTextBox == true && testgrainsSelectedFromComboBox == true && testgbSizeTextBox == true)
                        {
                                amount = Convert::ToInt32(gbAmountFromTextBox);
                                id = rand() % amount + 1;
                                for (int k = 0; k < amount; k++)
                                {
                                        for (int i = 0; i < rowTable - 1; i++)
                                        {
                                                for (int j = 0; j < colTable - 1; j++)
                                                {
                        if (table[i][j] != table[i][j + 1] && table[i][j] == id && table[i][j + 1] != id)
                                                        {
                                                                table[i][j] = 100;
                                                                tableGB[i][j] = table[i][j];
                                        board->DrawRectangle(pencil, i, j, gbSize, gbSize);
```

```
                                    }
                    if (table[i][j] != table[i + 1][j] && table[i][j] == id && table[i + 1][j] != id)
                                    {
                                        table[i][j] = 100;
                                        tableGB[i][j] = table[i][j];
                            board->DrawRectangle(pencil, i, j, gbSize, gbSize);
                                    }
                                }
                            }

                            for (int i = 1; i < rowTable - 1; i++)
                            {
                                for (int j = 1; j < colTable - 1; j++)
                                {
                    if (table[i][j] != table[i][j - 1] && table[i][j] == id && table[i][j - 1] != id)
                                    {
                                        table[i][j - 1] = 100;
                                    tableGB[i][j] = table[i][j - 1];
                            board->DrawRectangle(pencil, i, j, gbSize, gbSize);
                                    }
                    if (table[i][j] != table[i - 1][j] && table[i][j] == id && table[i - 1][j] != id)
                                    {
                                        table[i - 1][j] = 100;
                                    tableGB[i][j] = table[i - 1][j];
                            board->DrawRectangle(pencil, i, j, gbSize, gbSize);
                                    }
                                }
                            }
                            id = rand() % amount + 1;
                        }

                    }
                }
            }
}
```

```
private: System::Void clearSpaceButton_Click(System::Object^  sender, System::EventArgs^  e)
{
				board->Clear(Color::White);
				Pen ^ pencil = gcnew Pen(System::Drawing::Color::Navy);
				pencil->Color = System::Drawing::Color::Black;
				Graphics ^ picture = Graphics::FromImage(image1);

				for (int i = 0; i < rowTable; i++)
				{
					for (int j = 0; j < colTable; j++)
					{
						if (tableGB[i][j] == 100)
						{
							board->DrawRectangle(pencil, i, j, gbSize, gbSize);
						}
					}
				}
}

private: System::Void structureGenerateButton_Click(System::Object^  sender, System::EventArgs^  e)
{
				srand(time(NULL));
				bool testStructureComboBox = true;
				bool testAmountGrainFromDomainUpDown = true;
				int amountGrain = 0;
				int id = 0;

				structureFromComboBox = structureComboBox->Text;

				if (structureFromComboBox == " ")
				{
					MessageBox::Show("Options Structure Error. Please set!");
					testStructureComboBox = false;
				}
```

```
amountGrainFromDomainUpDown = amountGrainDomainUpDown->Text;

if (amountGrainFromDomainUpDown == " ")
{
        MessageBox::Show("Options Amount grain Error. Please set!");
        testAmountGrainFromDomainUpDown = false;
}

amountGrain = Convert::ToInt32(amountGrainFromDomainUpDown);

board = panel1->CreateGraphics();
Pen ^ pencil = gcnew Pen(System::Drawing::Color::Navy);
pencil->Color = System::Drawing::Color::Pink;
Graphics ^ picture = Graphics::FromImage(image1);

int tmpId = 0;
if (testStructureComboBox == true && testAmountGrainFromDomainUpDown == true)
{
        if (structureFromComboBox == "Substructure")
        {
                id = rand() % amountGrain + 1;
                tmpId = id;
                for (int k = 0; k < amountNucleon; k++)
                {
                        for (int i = 0; i < rowTable; i++)
                        {
                                for (int j = 0; j < colTable; j++)
                                {
                                        if (table[i][j] == id)
                                        {
                                                table[i][j] = tmpId;
                                        board->DrawRectangle(pencil, i, j, 1, 1);
                                        }
                                }
                        }
                        do
                        {
                            id = rand() % amountGrain + 1;
                        } while (id == tmpId);
                    }

                }
                else if (structureFromComboBox == "Dual Phase")
                {
                    id = rand() % amountGrain + 1;
                    int phase = 0;
                    for (int i = 0; i < amountNucleon + 1; i++)
                    {
                        if (tablePhase[i] > 1)
                        {
                            phase = tablePhase[i];
                            break;
                        }
                    }

                    for (int i = 0; i < rowTable; i++)
                    {
                        for (int j = 0; j < colTable; j++)
                        {
                            if (table[i][j] == phase)
                            {
                                table[i][j] = phase;
                                board->DrawRectangle(pencil, i, j, 1, 1);
                            }
                        }
                    }
                }
            }
}
```

```
private: System::Void grainsSelectedComboBox_SelectedIndexChanged(System::Object^  sender, System::EventArgs^  e) {

}

private: System::Void gbAmountTextBox_TextChanged(System::Object^  sender, System::EventArgs^  e) {

}

private: System::Void label1_Click(System::Object^  sender, System::EventArgs^  e) {

}

private: System::Void label2_Click(System::Object^  sender, System::EventArgs^  e) {

}

};

}
```

Source file:

```
#include "MyForm.h"


using namespace System;

using namespace System::Windows::Forms;


[STAThread]


void Main(array<String^>^ args)

{

	Application::EnableVisualStyles();

	Application::SetCompatibleTextRenderingDefault(false);

	simulationMultiscaleModelling::MyForm form;

	Application::Run(%form);

}
```