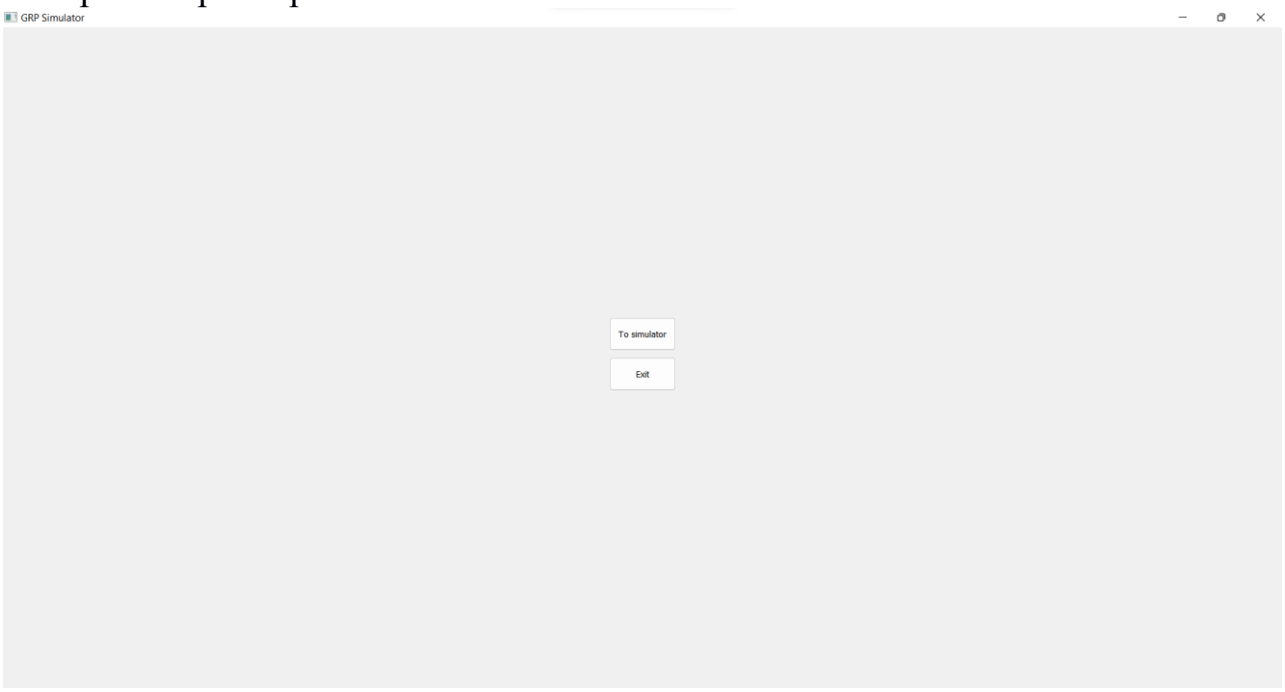
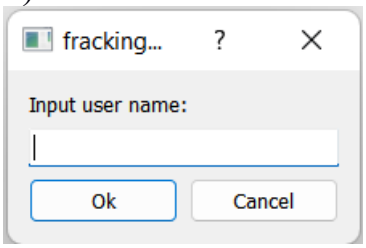
1. Постановка задачи
Реализовать программу симулятор влияния гидравлического разрыва пласта на добычу нефти. Программа должна обязательно включать в себя следующие элементы:
Задание названия месторождения через диалоговое окно, задание режима симулятора – вариантов месторождения, задание пользователем параметров месторождения и параметров ГРП, их учитывание в работе программы, использование графии – стандартных объектов, картинок и анимаций для визуального отображения результата ГРП (образовавшихся трещин), использование управления мышью, с обязательным считыванием координат, а также клавиатуры для изменения параметров ГРП, возможность сохранения картинки результата ГРП и итоговых параметров месторождения просле проведения Гидроразрыва пласта.

2. Инструкция к программной реализации
Программа Симулятор влияния гидравлического разрыва пласта на добычу нефти реализована для проведения исследований зависимости параметров месторождения после ГРП, от параметров месторождения и параметров ГРП. Программа позволяет также увидеть влияние Гидроразрыва пласта на скважину (появление трещин, показывающих успешность проведенной операции) в визуальном отображении программы.

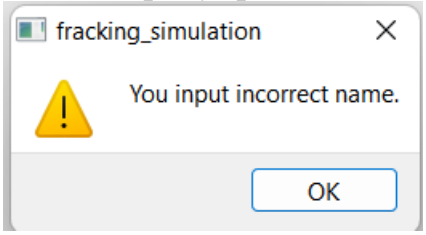
1) При запуске программы пользователю открывается Стартовая страница, на которой пользователь может выйти из программы, либо перейти к странице Настроек параметров ГРП:



2) После нажатия кнопки перехода к странице Настроек параметров ГРП



При попытке задания пустой строки в качетсве имени пользователя, программа выдает предупреждение:



3) После введения имени пользователя, он переходит на страницу Параметров ГРП.

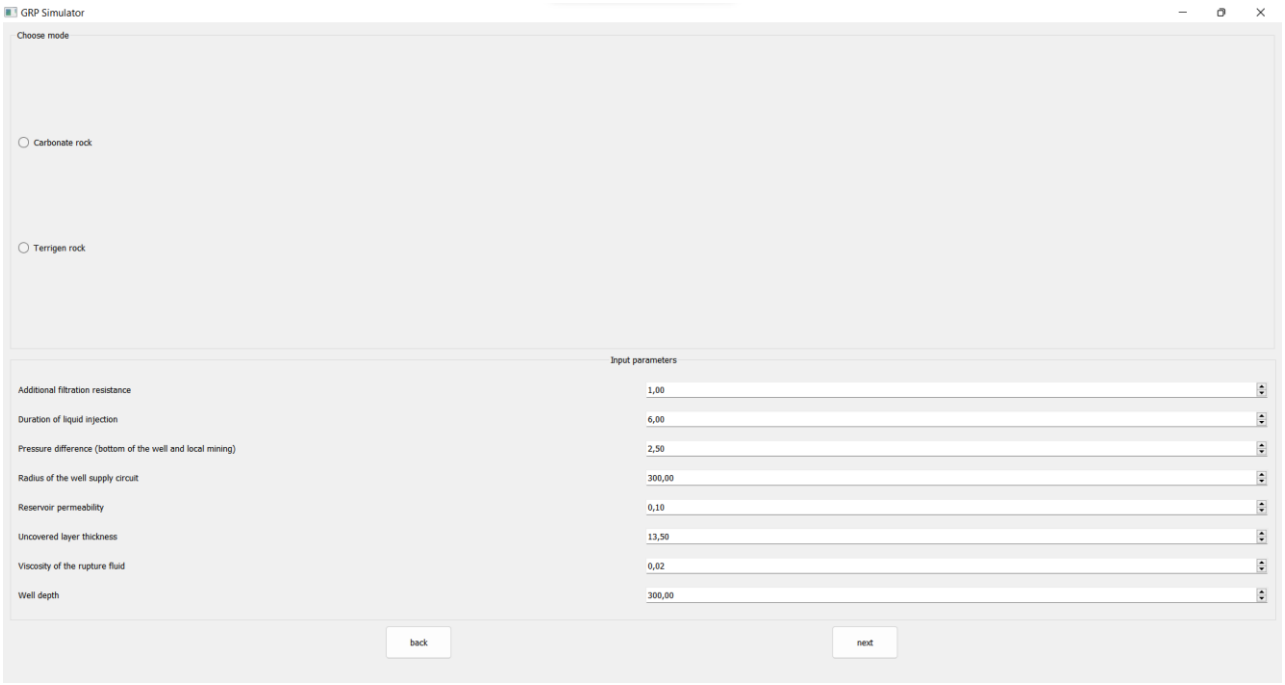
Здесь можно ввести следующие параметры месторождения:

1. Порода месторождения – карбонатная (для нее установлен модуль Юнга равен 7*10^4 Па, коэффициент Пуассона 0.3), или терригенная (модуль Юнга равен 3.5*10^4 Па, коэффициент Пуассона 0.13).

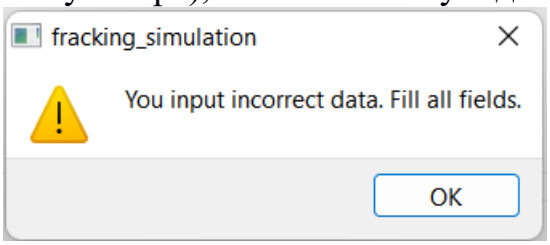
2. Параметры месторождения, скважины и параметры ГРП:

Дополнительное фильтрационное сопротивление скважины (0,1 – 10), длительность закачки жидкости (5-20 мин), превышение давления на забое скважины над локальным горным (2-4,2 МПа), радиус контура питания скважины (100-600 м), проницаемость пласта (0,01-1 мкм^2), вскрытая толщина пласта (8-20 м), вязкость жидкости разрыва (0,01-0,1 МПа/с), глубина скважины (300-3000 м).

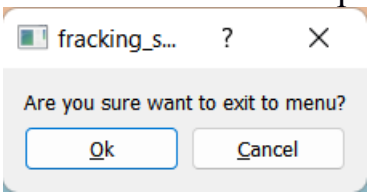
Изменять значения параметров можно по нажатию текстовое поле, содержащее текущий параметр и заданию другого значения, либо же с помощью «стрелок прокрутки», которые увеличивают значение параметра на определенный «шаг».



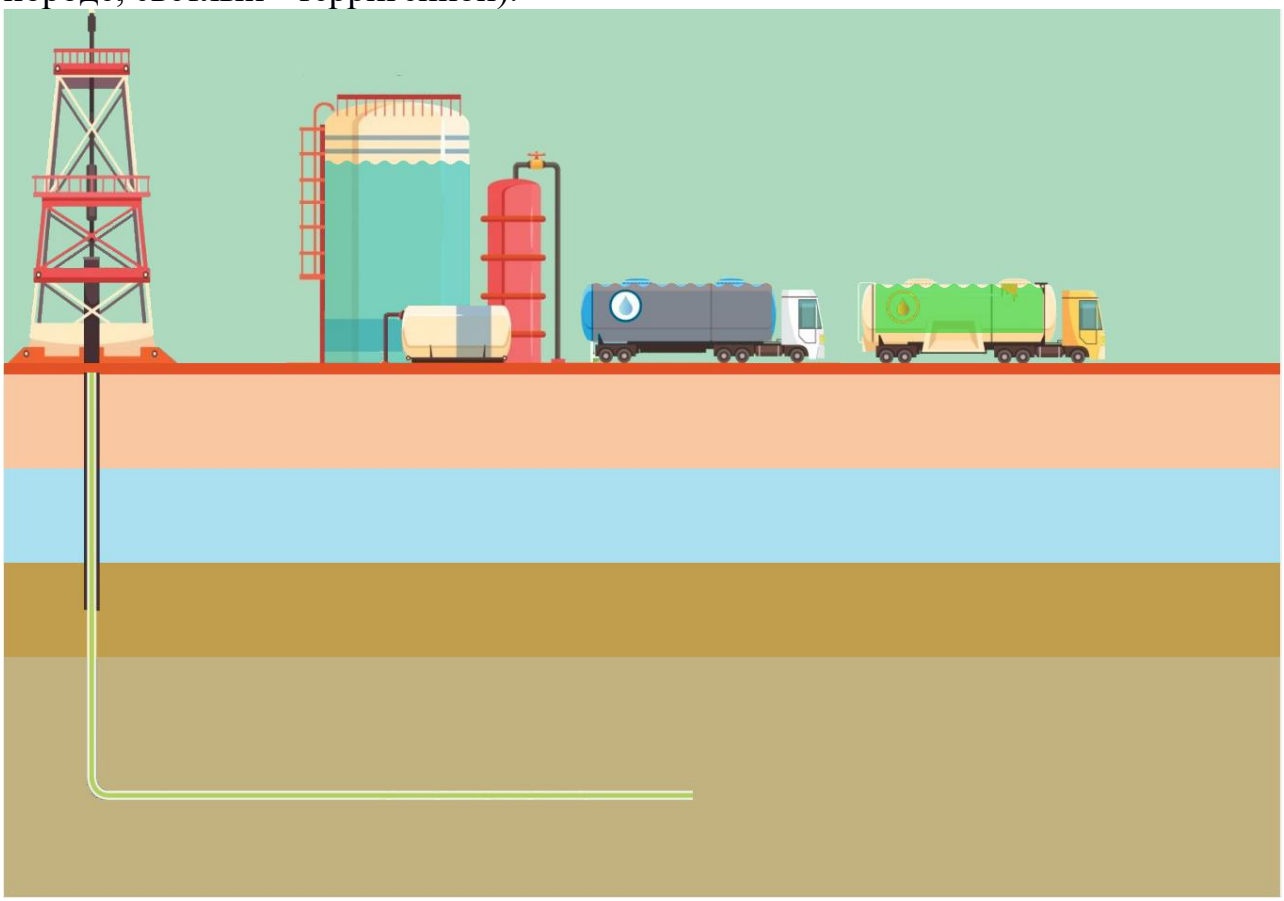
Пользователь может выйти на Стартовую страницу, нажав кнопку back, либо же после задания всех параметров может перейти к странице Графики. При попытке задания некорректных параметров (не введен режим работы симулятора), пользователь увидит предупреждение:



4) Страница Графики включает в себя кнопки выхода в меню и запуска процесса ГРП (о кнопке запуска ГРП будет сказано позже). При нажатии кнопки выхода в меню, всплывает уточняющая подсказка, где пользователь может остаться на странице Графики, либо выйти на Стартовую страницу:

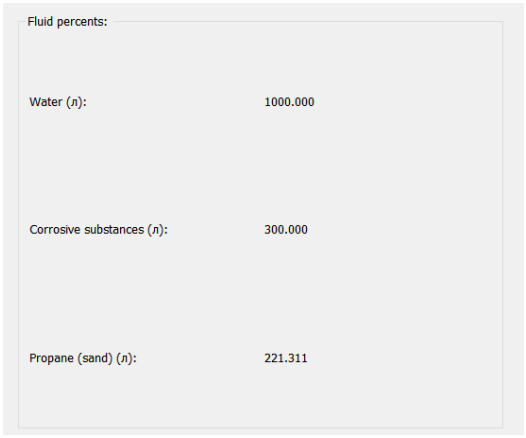


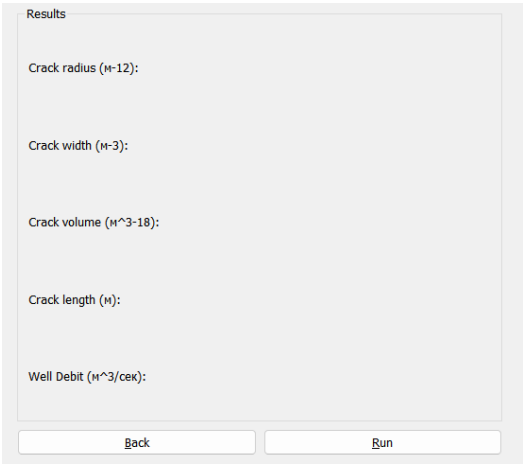
Кроме того страница представляет область Визуального отображения, где показаны: контейнеры, содержащие жидкости (резервуар и две автоцистерны), представлена установка для проведения ГРП, а также порода, залегающая выше коллектора и сам пласт-коллектор (в зависимости от режима симулятора, он будет разного цвета – более темный коричневый соответствует карбонатной породе, светлый - терригенной).



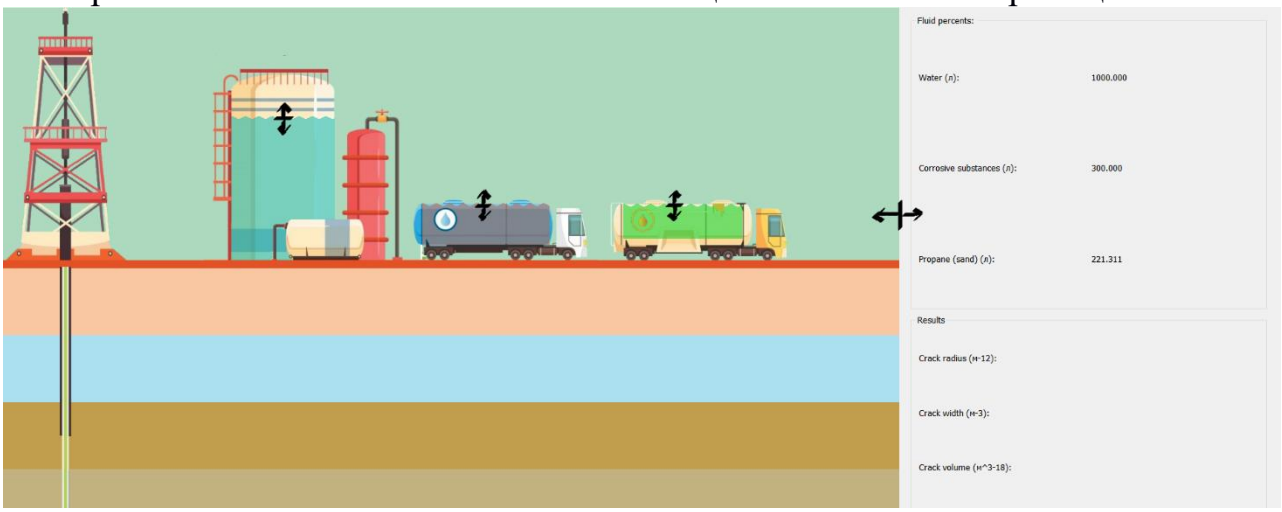
На области Визуального отображения можно менять уровень жидкостей разрыва, находящихся в резервуаре или в автоцистернах с помощью мыши

(«перетаскивая» уровень жидкости мышью, либо клавиатурой – кнопка «I» увеличит уровень жидкости в резервуаре на 96 (л), кнопка «D» - уменьшит уровень жидкости). По нажатии кнопки запуска ГРП, мы также сможем увидеть трещины в пласте, соответствующие итоговым данным проведения ГРП.
На странице Графики представлена также область показа текущих Параметров проведения ГРП (значения текущих объемов жидкостей разрыва и результирующие данные скважины после проведения ГРП, до нажатия кнопки запуска ГРП поля результирующих данных остаются пустыми).

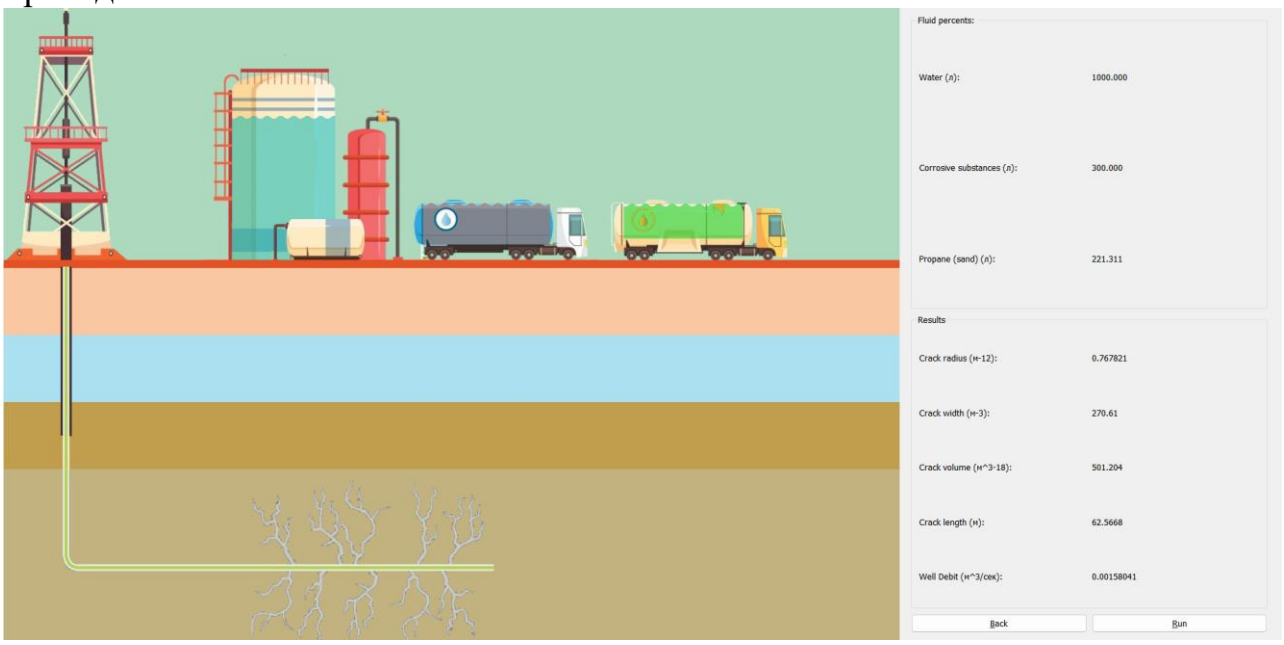




«Границу» между двумя областями можно «двигать» зажимая мышью.
На картинке помечены изменяемые с помощью мыши части страницы.

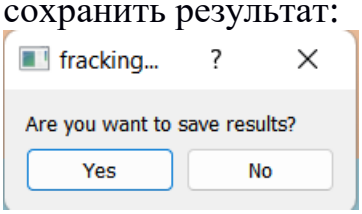


На следующей картинке представлена страница, после нажатия кнопки «Run» - проведения ГРП:



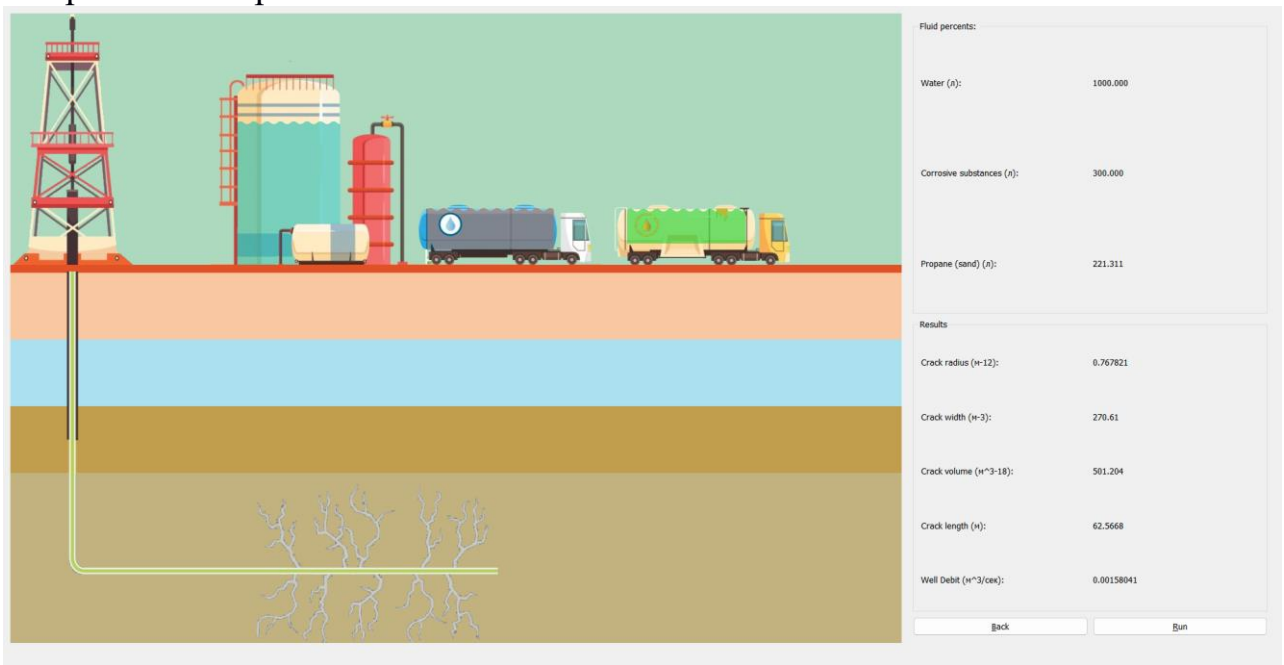
Пользователю предоставляется графическое отображения результата Гидроразрыва пласта в виде образовавшихся трещин, а также характеристики скважины после проведения ГРП в области Параметров ГРП.

5) Сразу после проведения симуляционного ГРП, пользователю предлагается сохранить результат:



Если пользователь хочет сохранить результат – он должен выбрать во всплывающем окне путь, по которому сохранится картинка с визуальным отображением результата ГРП и pdf файл с информацией о проведенном ГРП.

Сохраненная картинка:



Текст файла pdf с результатом:

User parameters:
User name: 2
Mode: Terrigen
Well depth(м): 300
Reservoir permeability (мкм^2): 0.1
Viscosity of the rupture fluid (МПа/с): 0.02
Radius of the well supply circuit (м): 300
Pressure difference (bottom of the well and local mining) (МПа): 2.5
Uncovered layer thickness (м): 13.5
Additional filtration resistance: 1
Result parameters:
New well debit (м^3/сек): 0.00158041
Crack radius (м-12): 0.767821
Crack width (м-3): 270.61
Crack volume (м^3-18): 501.204
Crack length (м): 62.5668

После сохранения, пользователю предлагают выйти в Стартовое меню.

3.Программный код и пояснения к нему.
В программе есть следующие классы:
MainWindow, StartWidget, FrackingSettingsWidget, Parameters.h, InterfaceGraphicsHandler, GraphicsWidget и GrahicsParametersWidget.
StartWidget – виджет Стартовой страницы, содержит только две кнопки, позволяющие либо выйти, либо идти к странице Настроек:
Настройки компоновщика:

```
StartWidget::StartWidget(QWidget *parent)
    : QWidget{parent}
{
    resize(1920,1080);

    btn_to_simulator_settings = new QPushButton("&To simulator", this);
    btn_to_simulator_settings->setFixedSize(100, 50);
    btn_to_simulator_settings->show();
    btn_to_simulator_settings->move(this->width()/2-50, this->height()/2-105);
    btn_exit = new QPushButton("&Exit", this);
    btn_exit->show();
    btn_exit->setFixedSize(100, 50);
    btn_exit->move(this->width()/2-50, this->height()/2-45);
}
```

Подключение кнопок «выйти» и идти дальше:

```
    QObject::connect(startWidget->btn_to_simulator_settings, SIGNAL(clicked()), this, SLOT(setSimulatorSettingsWidget()));
    QObject::connect(startWidget->btn_exit, SIGNAL(clicked()), this, SLOT(slotClose()));
```

Эти слоты разобраны далее.

FrackingSettingsWidget, виджет страницы Настроек, содержит внутри себя экземпляр класса Parameters. В программе класс Parameters выполняет функции хренения введенных пользователем параметров, также содержит виджеты (SpinBox-ы и Label-ы) использующие для вводом пользователя параметров. Также все функции расчета ГРП содержатся в классе Parameters.
Пример задания настроек полей для ввода параметров:

```
    mapped_parameters_line_edits["Viscosity of the rupture fluid"]=edit_m;
    edit_m->setRange(0.01,0.1);
    edit_m->setSingleStep(0.01);
    mapped_parameters_line_edits["Duration of liquid injection"]=edit_t;
    edit_t->setRange(5,20);
    edit_t->setSingleStep(1);
```

Далее представлены эти функции:

```
double Parameters::getChackLength(){
    double ChackLength = sqrt(100*LiquidVolume*UngE/(5.6*(1-n)*(1-n)*H*delta_P));
    return ChackLength;
}

double Parameters::getChackRadius(){
    double C = 0.0269-3.21*L/1000000;
    double rc = C*sqrt(give_Q*sqrt(m*t/K));
    return rc;
}

double Parameters::getChackWidth(){
    double chack_length = getChackLength();
    double w = 8*(1-n)*(1-n)*delta_P*chack_length/UngE;
    return w;
}
double Parameters::getChackVolume(){
    double Vt = getChackWidth()*getChackRadius()*getChackRadius()*M_PI;
    return Vt;
}

double Parameters::getNewDebit(){
    double Qt = 2*M_PI*K*10*delta_P/(100000*m*qLn(Rk*1000/LiquidVolume*getChackRadius()));
    return Qt;
}

double Parameters::getGRPEffect(){
    double Ef = getNewDebit()/getDebitBeforeGRP();
    return Ef;
}

double Parameters::getDebitBeforeGRP(){
    return getNewDebit()*qLn(Rk/getChackRadius())/(C_dop+qLn(Rk*LiquidVolume/1000/getChackRadius()));
}
```

Программа выполнена с использованием класса MainWindow.h (наследником QMainWindow). В программе этот класс выполняет роль главного управляющего класса: управляет переключениями виджетов (которое реализовано с помощью QStackedWidget), управляет вызовом Диалоговых окон, содержит перегрузку KeyPressEvent для изменения уровня жидкости в резервуаре на странице Графики по нажатию клавиатуры, управляет сохранением картинки и pdf файла результатов ГРП, а также реализует логику передачи информации между Виджетами с помощью сигналов и слотов.
MainWindow.h:

```
19  class MainWindow : public QMainWindow
20  {
21      Q_OBJECT
22
23  public:
24      MainWindow(QWidget *parent = nullptr);
25      ~MainWindow();
26      QStackedWidget *stackedWidget;
27      StartWidget *startWidget;
28      ScoresWidget *scoresWidget;
29      Interface_graphics_handler *interface_graphics_handler ;
30  //    SessionHandler *sessionHandler;
31      QDialog *input_name_dialog;
32      QLineEdit *input_name_dialog_text_edit;
33
34  //    QWidget *fracking_settings_widget;
35      FrackingSettingsWidget *frackingSettingsWidget;
36
37      QPixmap original_pixmap;
38
39      void saveScreenshot();
40      void takeScreenshot();
41      void saveResultFile();
42  protected:
43
44      virtual void keyPressEvent(QKeyEvent *event);
```

```
46  public slots:
47      void setScoresWidget();
48      void setSimulatorSettingsWidget();
49      void setStartWidget();
50      void setGraphicsWidget();
51      void backToStartWidget();
52
53      void slotDialogScreenshot();
54
55      void accept_if_valid_dialog_name();
56      void slotClose();
57
58  //signals:
59  //    void signalWaveFluidIncrease();
60  //    void signalWaveFluidDecrease();
61  private:
62      Ui::MainWindow *ui;
63      QString user_name;
64      QString file_dir;
65  //    Parameters *user_parameters;
66
67  };
```

Прописаны объявления всех виджетов, использующихся в программе, объявление Диалоговых окон input_name_dialog, картинка для сохранения

original_pixmap, слоты переключения виджетов (например setScoresWidget() выглядит так)

```
100 void MainWindow::setScoresWidget()
101 {
102     stackedWidget->setCurrentIndex(Widgets::scores_widget);
103 }
```

Аналогично выглядят и остальные, но в некоторых добавляется проверка диалогового окна (переключение на виджет симулятора работает только после диалогового окна на ввод имени):

```
105 void MainWindow::setSimulatorSettingsWidget()
106 {
107     if(input_name_dialog->exec()== QDialog::Accepted){
108         this->user_name = input_name_dialog_text_edit->text();
109         stackedWidget->setCurrentIndex(Widgets::fracking_settings_widget);
110     }
111
112 }
```

В MainWindow.h определены также функции saveScreenshot(), takeScreenshot() и SaveResultFile():
Захват картинки экрана происходит при помощи функции QWidget.grab()

```
253 void MainWindow::takeScreenshot()
254 {
255     original_pixmap = this->grab(this->rect());
256 }
```

Сохранение полученного скриншота происходит следующим образом:
С помощью Диалогового окна QFileDialog получаем от пользователя путь к картинке, а с помощью функции original_pixmap.save(fileName) сохраняем по введенному пути картинку-скриншот.

```
224 void MainWindow::saveScreenshot()
225 {
226     const QString format = "png";
227     QString initialPath = QStandardPaths::writableLocation(QStandardPaths::PicturesLocation);
228     if (initialPath.isEmpty())
229     initialPath = QDir::currentPath();
230     initialPath += tr("/untitled.") + format;
231
232     QFileDialog fileDialog(this, tr("Save As"), initialPath);
233     fileDialog.setAcceptMode(QFileDialog::AcceptSave);
234     fileDialog.setFileMode(QFileDialog::AnyFile);
235     fileDialog.setDirectory(initialPath);
236     QStringList mimeTypes;
237     const QList<QByteArray> baMimeTypes = QImageWriter::supportedMimeTypes();
238     for (const QByteArray &bf : baMimeTypes)
239     mimeTypes.append(QLatin1String(bf));
240     fileDialog.setMimeTypeFilters(mimeTypes);
241     fileDialog.selectMimeTypeFilter("image/" + format);
242     fileDialog.setDefaultSuffix(format);
243     if (fileDialog.exec() != QDialog::Accepted)
244         return;
245     const QString fileName = fileDialog.selectedFiles().first();
246     file_dir = fileName;
247     if (!original_pixmap.save(fileName)) {
248         QMessageBox::warning(this, tr("Save Error"), tr("The image could not be saved to \"%1\".")
249             .arg(QDir::toNativeSeparators(fileName)));
250     }
251 }
```

Функция saveResultsFile() получает из виджета управления графикой interface_graphics_handler параметры-результаты ГРП, а от объекта

frackingSettingsWidget начальные параметры симулятора.

```
void MainWindow::saveResultFile(){
    interface_graphics_handler->parameters_widget->result_params;                    Expression resu
    QString str;
    str += "User parameters: \n";
    str += "User name: " + user_name+"\n";
    str += "Mode: "+QString(frackingSettingsWidget->parameters->getMode()==0?"Carbonate":"Terrigen")+"\n";
    str += "Well depth(м): " + QString::number(frackingSettingsWidget->parameters->L)+"\n";
    str += "Reservoir permeability (мкм^2): " + QString::number(frackingSettingsWidget->parameters->K)+"\n";
    str += "Viscosity of the rupture fluid (МПа/с): " + QString::number(frackingSettingsWidget->parameters->m)+"\n";
    str += "Radius of the well supply circuit (м): " + QString::number(frackingSettingsWidget->parameters->Rk)+"\n";
    str += "Pressure difference (bottom of the well and local mining) (МПа): " + QString::number(frackingSettingsWidget->parameters->delta_P)+"\n";
    str += "Uncovered layer thickness (м): " + QString::number(frackingSettingsWidget->parameters->H)+"\n";
    str += "Additional filtration resistance: " + QString::number(frackingSettingsWidget->parameters->C_dop)+"\n";

    str += "Result parameters:\n";
    str += "New well debit (м^3/сек): " + interface_graphics_handler->parameters_widget->result_params["debit"]+"\n";
    str += "Crack radius (м-12): " + interface_graphics_handler->parameters_widget->result_params["rad"]+"\n";
    str += "Crack width (м-3): " + interface_graphics_handler->parameters_widget->result_params["width"]+"\n";
    str += "Crack volume (м^3-18): " + interface_graphics_handler->parameters_widget->result_params["volume"]+"\n";
    str += "Crack length (м): " + interface_graphics_handler->parameters_widget->result_params["length"]+"\n";

    QString directory_to_save = file_dir.left(file_dir.length()-4)+".pdf";
    qDebug()<<directory_to_save;
    QTextDocument *tmp_document = new QTextDocument(str);
    QPrinter printer(QPrinter::HighResolution);
    printer.setOutputFormat(QPrinter::PdfFormat);
    printer.setOutputFileName((directory_to_save));
    tmp_document->print(&printer);


}
```

Код (MainWindow.cpp, конструктор MainWindow::MainWindow()), отображающий управление Виджетами и передачу между ними информации:

```
    stackedWidget = new QStackedWidget(this);


//    startWidget
    startWidget = new StartWidget(stackedWidget);
    QObject::connect(startWidget->btn_to_simulator_settings, SIGNAL(clicked()), this, SLOT(setSimulatorSettingsWidget()));
    QObject::connect(startWidget->btn_exit, SIGNAL(clicked()), this, SLOT(slotClose()));

    stackedWidget->addWidget(startWidget);

//  fracking_settings widget
    frackingSettingsWidget = new FrackingSettingsWidget(this);
    QObject::connect(frackingSettingsWidget->button_back, SIGNAL(clicked()), this, SLOT(setStartWidget()));
    QObject::connect(frackingSettingsWidget->button_next, SIGNAL(clicked()), this, SLOT(setGraphicsWidget()));
    stackedWidget->addWidget(frackingSettingsWidget);

//    main_widget:
    interface_graphics_handler = new Interface_graphics_handler(this);
    stackedWidget->addWidget(interface_graphics_handler);
    QObject::connect(interface_graphics_handler->parameters_widget->button_back, SIGNAL(clicked()), this, SLOT(backToStartWidget()));
    QObject::connect(interface_graphics_handler, SIGNAL(signalAskResultsOfGrp(int)), frackingSettingsWidget, SLOT(getResultsOfGrp(int)));
    QObject::connect(frackingSettingsWidget,SIGNAL(translateResultsOfGrp(double,double,double,double,double)),
                     interface_graphics_handler, SLOT(receiveResultsOfGrp(double,double,double,double,double)));
    QObject::connect(interface_graphics_handler, SIGNAL(signalDialogScreenshot()), this, SLOT(slotDialogScreenshot()));

//    out
    setCentralWidget(stackedWidget);
```

В этом фрагменте происходит добавление виждетов: Стартовый, виджет Настройки параметров, виджет управления Графикой.
Большинство описанных на картинке connect’ов сигналов и слотов нужны для связи нажатий кнопок и смены Виджетов, рассмотрим следующие сигналы:

```
QObject::connect(interface_graphics_handler, SIGNAL(signalAskResultsOfGrp(int)), frackingSettingsWidget, SLOT(getResultsOfGrp(int)));
```

Сигнал с аргументом для получения результатов ГРП нужен при нажатии на кнопку запуска ГРП, в качестве параметра идет объем жидкости – виджет interface_graphics_handler «знает этот параметр». Слот записывает в переменные класса Parameters параметр LiquidVolume:

```
void FrackingSettingsWidget::getResultsOfGrp(int liquidVolume)
{
    parameters->setLiquidVolume(liquidVolume);
    double  ChackLength =parameters->getChackLength();
    double  Qt=parameters->getNewDebit();
    double  Vt=parameters->getChackVolume();
    double  w=parameters->getChackWidth();
    double  rc=parameters->getChackRadius();
    emit translateResultsOfGrp(ChackLength,
                               Qt,
                               Vt,
                               w,
                               rc);
}
```

То есть происходит вычисление результирующих параметров, так как функции для их вычисления лежат только в объекте класса Parameters.h, объект которого есть только в Settings виджете.
Далее с помощью следующей связи результирующие данные передаются обратно в interface_graphics_handler:

```
42      QObject::connect(frackingSettingsWidget,SIGNAL(translateResultsOfGrp(double,double,double,double,double)),
43                       interface_graphics_handler, SLOT(receiveResultsOfGrp(double,double,double,double,double)));
```

Слот «расставляет» в области показа результирующих параметров соответствующие данные и вызывает функцию slotActivatedGRP, отвечающую за изменение размера трещин по имеющимся параметрам:

```
67  void Interface_graphics_handler::receiveResultsOfGrp(double ChackLength, double Qt, double Vt, double w, double rc)
68  {
69      parameters_widget->result_params["rad"] =QString::number(rc);
70      parameters_widget->result_params["width"] =QString::number(w);
71      parameters_widget->result_params ["volume"] =QString::number(Vt) ;
72      parameters_widget->result_params ["length"] =QString::number(ChackLength) ;
73      parameters_widget->result_params["debit"] =QString::number(Qt);
74
75      parameters_widget->crack_radius_label_score->setText(QString::number(rc));
76      parameters_widget->crack_width_label_score->setText(QString::number(w));
77      parameters_widget->crack_volume_label_score->setText(QString::number(Vt));
78      parameters_widget->crack_length_label_score->setText(QString::number(ChackLength));
79      parameters_widget->crack_debit_label_score->setText(QString::number(Qt));
80
81      qDebug()<< "Default_Values" <<w<<ChackLength<<rc<<w*ChackLength*rc;
82
83      graphics_widget->slotActivatedGRP((double)LiquidVolume/(double)average_volume);
84  }
```

Трещина увеличивается/уменьшается с переданным множителем.

```
303 void GraphicsWidget::slotActivatedGRP(double scale)
304 {
305     qDebug()<<"SLOT ACTIVATED GRP";
306     activated_grp = true;
307     this->scale = scale;
308     for(int i = 1; i <= 10; i++){
309         QString cur_way = QString(images_path+"/cracks/crack_%1.png").arg(QString::number(i));
310         images[QString("crack_%1").arg(QString::number(i))]= QPixmap();
311         images[QString("crack_%1").arg(QString::number(i))].load(cur_way);
312     }
313     for(int i = 1; i <= 10; i++){
314         QString cur_name = QString("crack_%1").arg(QString::number(i));
315 //        qDebug()<<images[cur_name].size();
316         double size_before_w = (double)images[cur_name].width();
317         double size_after_w = (double)images[cur_name].width()*scale;
318         double size_before_h = (double)images[cur_name].height();
319         double size_after_h = (double)images[cur_name].height()*scale;
320         images[cur_name]=images[cur_name].scaled(size_after_w,size_after_h,Qt::IgnoreAspectRatio);
321 //        qDebug()<<images[cur_name].size();
322     }
323 }
```

Страница Графики реализована с помощью класса InterfaceGraphicsHandler который объединяет GrahicsParametersWidget и GraphicsWidget.
InterfaceGraphicsHandler разделяет эти два виджета с помощью Splitter-а:
InterfaceGraphicsHandler.h

```
QSplitter *splitter = new QSplitter(parent);
splitter->setChildrenCollapsible(true);

splitter->addWidget(graphics_widget);
splitter->addWidget(parameters_widget);


QVBoxLayout *containter = new QVBoxLayout;
containter->addWidget(splitter);
setLayout(containter);
```

Класс GraphicsWidget – прорисовывает каждый кадр с помощью QTimer и QPainter.
С помощью QPainter идет загрузка изображений, каждое изображение записывается в QMap<QString, QPixmap> images:

```
    images["truck_1"] = QPixmap();
    images["truck_1"].load(images_path+"car1.png");
    images["truck_2"] = QPixmap();
    images["truck_2"].load(images_path+"car2.png");
    images["derick"] = QPixmap();
    images["derick"].load(images_path+"derick.png");
    images["reservoir"] = QPixmap();
    images["reservoir"].load(images_path+"/reservoir.png");
    images["reservoirs_not_active"] = QPixmap();
    images["reservoirs_not_active"].load(images_path+"reservoirs_not_active.png");
    images["truck_1_base"] = QPixmap();
    images["truck_1_base"].load(images_path+"car1_base.png");
    images["truck_2_base"] = QPixmap();
    images["truck_2_base"].load(images_path+"car2_base2.png");
    images["well"] = QPixmap();
    images["well"].load(images_path+"well.png");
```

В перегрузке функции PaintEvent происходит прорисовка загруженных изображений по заданным координатам (содержатся в images_coords):

```
    for(QMap<QString, QPoint>::Iterator iter = images_coords.begin(); iter!= images_coords.end(); iter++){
        QString cur_name = iter.key();
        painter.drawPixmap(iter.value(),images[cur_name]);
    }
```

Также в программе используется класс Wave, для прорисовки жидкости в резервуарах.

```
class Wave{
private:
    QPainterPath *wave_frame;
    QPainter *painter;
public:
    QRect *max_characteristics;
    QRect *characteristics;
    double percent = 1.0;
    double amplitude;
    double w;
    double result_offset;
    double m_offset;
    double keyboard_velocity = 0.02;
```

Объект класса Wave прорисовывает текущий «кадр» волны с помощью функции draw_frame()

```
    void draw_frame(){
        wave_frame->moveTo(characteristics->x(), characteristics->y()+characteristics->height());
        for (int i = 0; i <= characteristics->width(); i+=1) // x изменяется от значения 0 ~ w до синусоиды
        {
            double waveY = (double) (amplitude * sin (w * i + result_offset)) ; // waveY изменяется при изменении значения x
            wave_frame->lineTo (characteristics->x()+i, waveY+characteristics->y()); // Рисуем линию от последней точки рисо
        }
        wave_frame->lineTo (characteristics->width()+characteristics->x(), characteristics->y()+characteristics->height());
        painter->drawPath(*wave_frame);
        result_offset+=m_offset;
    }
```

Класс GraphicsParametersWidget показывает текущее количество жидкостей разрыва в каждом из контейнеров, а также результаты проведения ГРП:

```
graphics_parameters_widget::graphics_parameters_widget(QWidget *parent)
    : QWidget{parent}
{
    water_level = new QLabel(this);
    corrosive_substances_level = new QLabel(this);
    propane_level = new QLabel(this);

    QGroupBox *groupBoxFluids = new QGroupBox(tr("Fluid percents: "));

    QGridLayout*insertedGridLayout = new QGridLayout();
    QLabel *tmp_label = new QLabel(this);
    tmp_label->setText("Water (n):");
    insertedGridLayout->addWidget(tmp_label,0,0);
    insertedGridLayout->addWidget(water_level,0,1);
    QLabel *tmp_label_2 = new QLabel(this);
    tmp_label_2->setText("Corrosive substances (n):"); //разъедающие
    insertedGridLayout->addWidget(tmp_label_2,1,0);
    insertedGridLayout->addWidget(propane_level,1,1);
    QLabel *tmp_label_3 = new QLabel(this);
    tmp_label_3->setText("Propane (sand) (n):"); //пропант
    insertedGridLayout->addWidget(tmp_label_3,2,0);
    insertedGridLayout->addWidget(corrosive_substances_level,2,1);
    groupBoxFluids->setLayout(insertedGridLayout);
```

```
    QGridLayout*insertedGridResults = new QGridLayout();
    QLabel * crack_radius_label= new QLabel(this);
    crack_radius_label->setText("Crack radius (м-12):");
    insertedGridResults->addWidget(crack_radius_label,0,0);
    insertedGridResults->addWidget(crack_radius_label_score,0,1);
    QLabel * crack_width_label = new QLabel(this);
    crack_width_label->setText("Crack width (м-3):"); //разъедающие
    insertedGridResults->addWidget(crack_width_label,1,0);
    insertedGridResults->addWidget(crack_width_label_score,1,1);
    QLabel * crack_volume_label = new QLabel(this);
    crack_volume_label->setText("Crack volume (м^3-18):"); //пропант
    insertedGridResults->addWidget(crack_volume_label,2,0);
    insertedGridResults->addWidget(crack_volume_label_score,2,1);
    QLabel * crack_length_label= new QLabel(this);
    crack_length_label->setText("Crack length (м):"); //пропант
    insertedGridResults->addWidget(crack_length_label,3,0);
    insertedGridResults->addWidget(crack_length_label_score,3,1);
    QLabel * crack_debit_label= new QLabel(this);
    crack_debit_label->setText("Well Debit (м^3/сек):"); //пропант
    insertedGridResults->addWidget(crack_debit_label,4,0);
    insertedGridResults->addWidget(crack_debit_label_score,4,1);
    groupBoxResults->setLayout(insertedGridResults);
```

Пример работы изменения уровня жидкости в контейнере и соответствующего отображения в GraphicsParametersWidget:
InterfaceGraphicsHandler конструктор (аргументы следующие: QString содержит информацию в каком именно контейнере поменялся уровень, второй аргумент содержит новое значение уровня жидкости):

```
QObject::connect(graphics_widget,SIGNAL(changedFluidLevel(QString, double)),parameters_widget, SLOT(slotChangedFluidLevel(QString, double)));
```

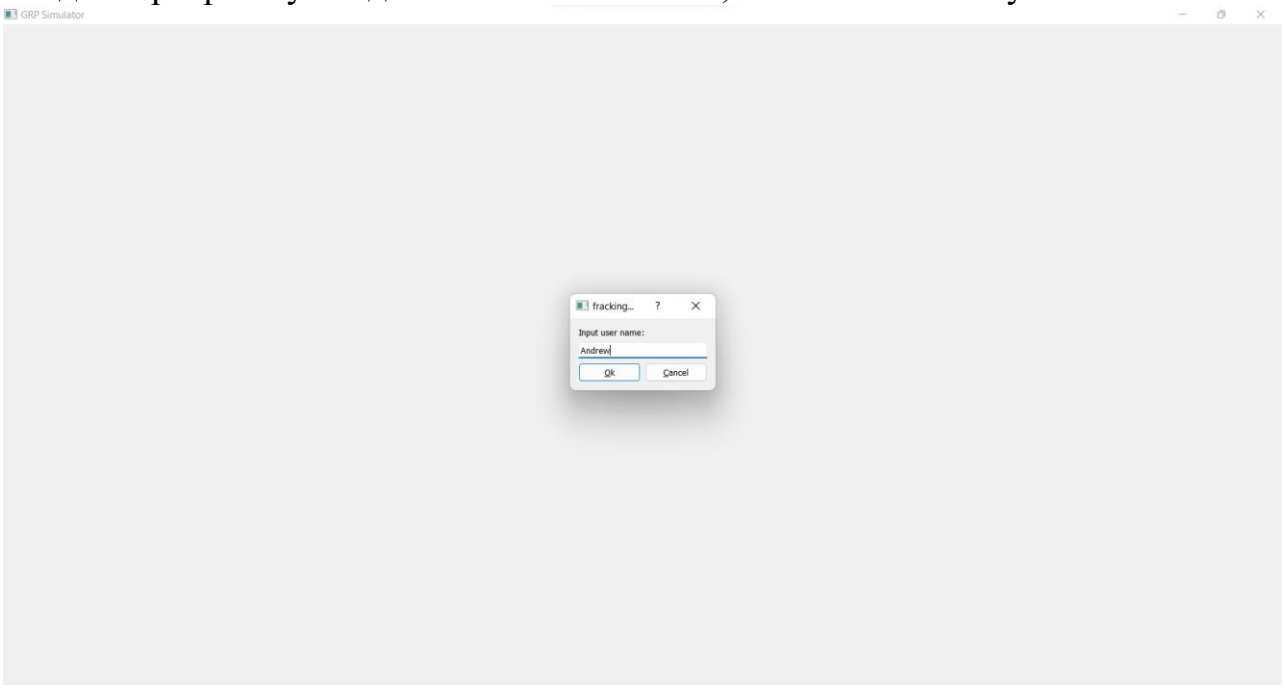
При изменении мышью уровня жидкости происходит вызов сигнала (wave_fluid – это QMap<QString, Wave> где хранятся все объекты контейнеров для жидкости – Резервуар, и две автоцистерны)

```
void GraphicsWidget::mouseMoveEvent(QMouseEvent *pe)
{
    if(wave_fluid["reservoir"]->max_characteristics->contains(pe->pos())){
        fluids_relative_coords["reservoir"].setY(pe->y()-images_coords["reservoir"].y());
        emit changedFluidLevel("reservoir",wave_fluid["reservoir"]->get_calculated_percent());
    } else if(wave_fluid["truck_1"]->max_characteristics->contains(pe->pos())){
        fluids_relative_coords["truck_1"].setY(pe->y()-images_coords["truck_1"].y());
        emit changedFluidLevel("truck_1",wave_fluid["truck_1"]->get_calculated_percent());
    } else if(wave_fluid["truck_2"]->max_characteristics->contains(pe->pos())){
        fluids_relative_coords["truck_2"].setY(pe->y()-images_coords["truck_2"].y());
        emit changedFluidLevel("truck_2",wave_fluid["truck_2"]->get_calculated_percent());
    }
}
```

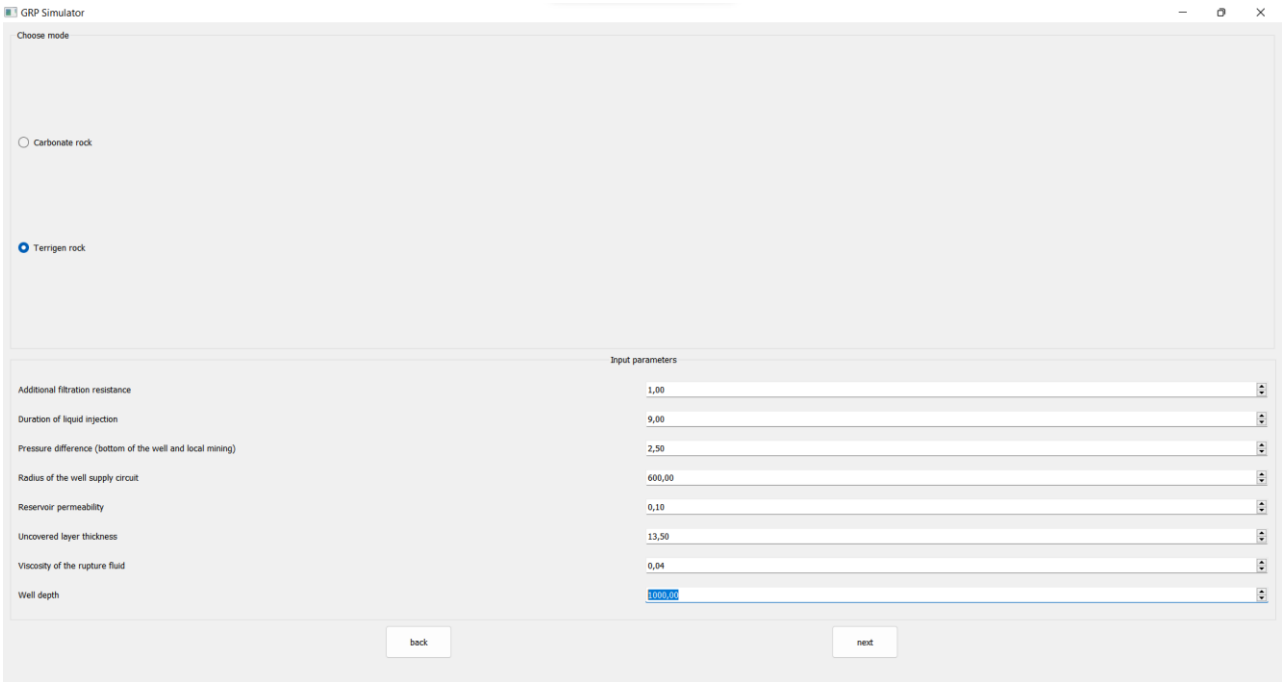
Слот изменения уровня жидкости – записывает новые данные на виджет показа параметров:

```
void graphics_parameters_widget::slotChangedFluidLevel(QString str, double level)
{
    if(str=="reservoir"){
        water_level->setText(QString::number(level*fluid_volume_water,'f',3));
    } else if(str == "truck_1"){

        corrosive_substances_level->setText(QString::number(level*fluid_volume_propane,'f',3));
    } else {

        propane_level->setText(QString::number(level*fluid_volume_corrosive,'f',3));
    }
}
```

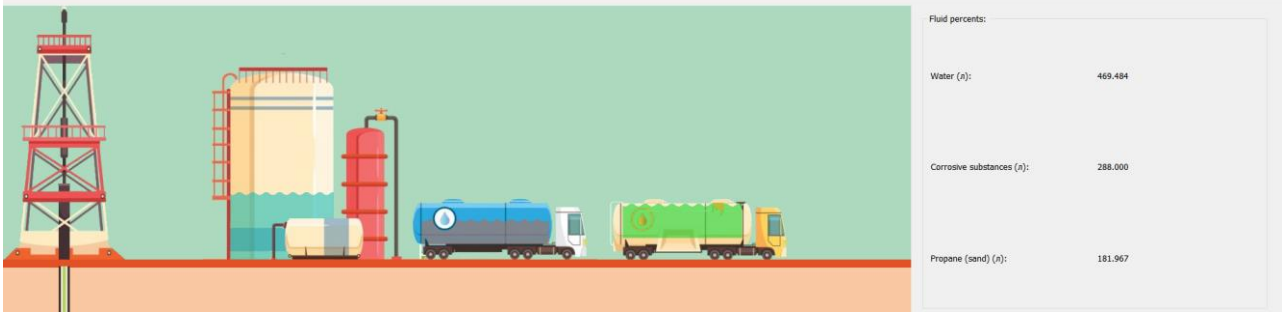
4. Демонстрация работы программы (скриншоты) с описанием
Зайдя в программу вводим имя пользователя, нажимаем кнопку «Ok»:



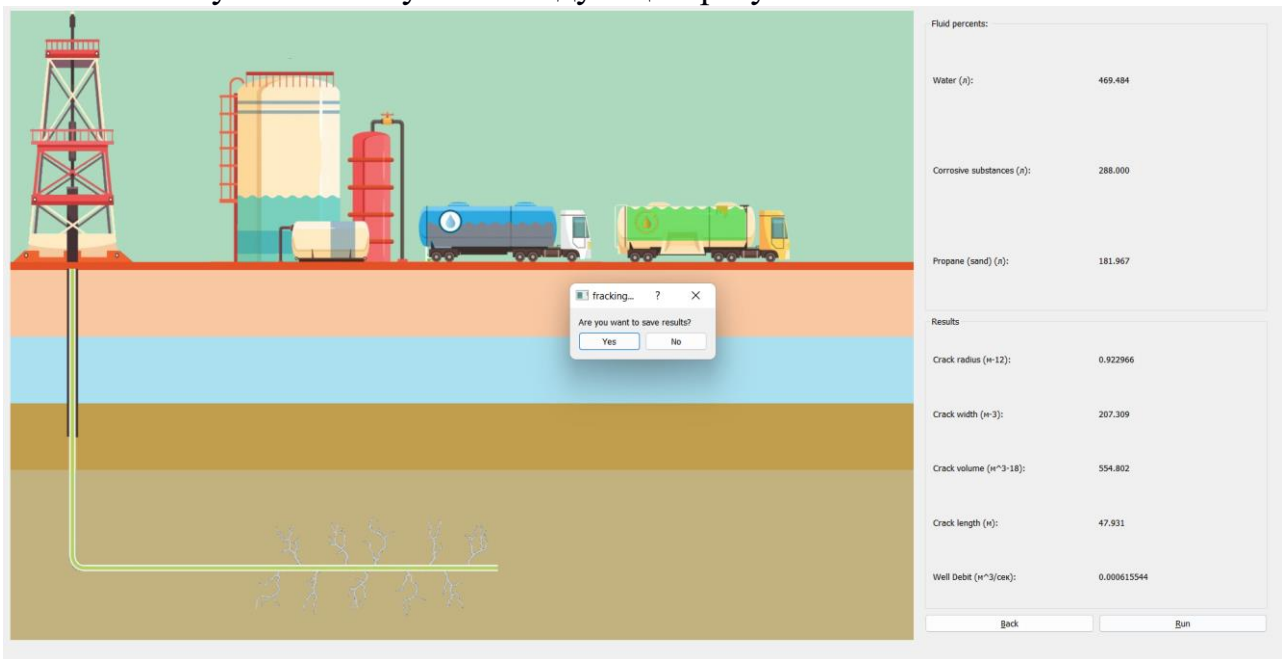
Задаем параметры месторождения, скважины и ГРП, изменив тип коллектора на терригенный, задав остальные параметры следующим образом:



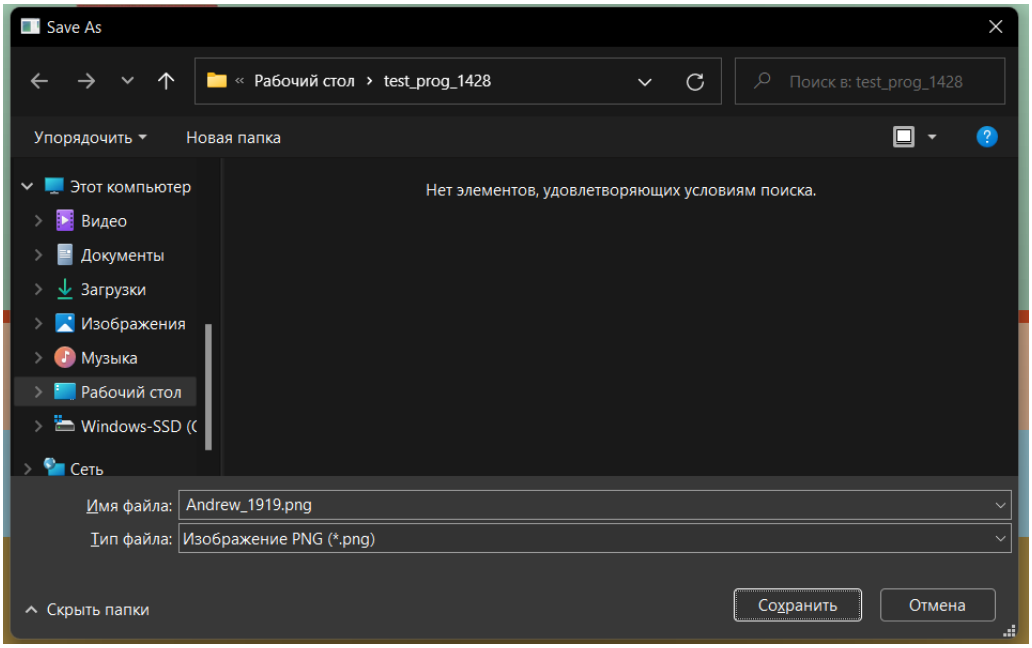
Нажимаем кнопку “next”.
Меняем объем используемой жидкости с помощью мыши на 470, 288 и 181 литр соответственно:



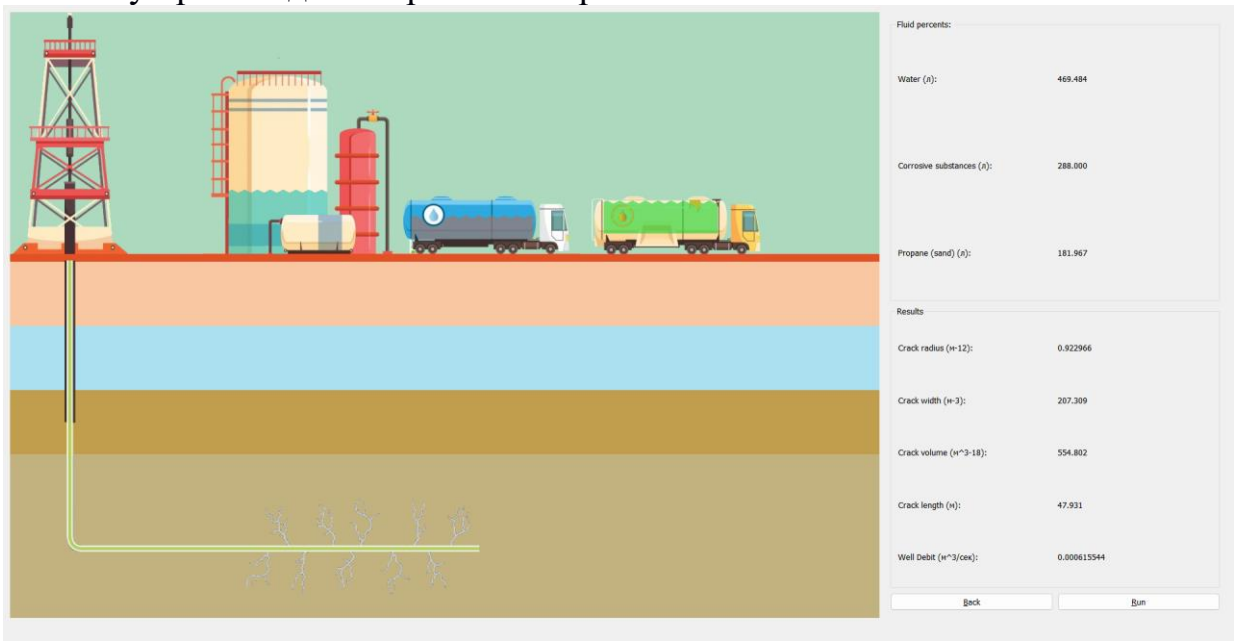
Нажав кнопку “Run” получаем следующий результат:



Мы видим получившиеся параметры Гидроразрыва и трещины, показывающие успешность ГРП.
Нажав кнопку Yes сохраняем результат в следующую дирректорию под названием:



По итогу происходит сохранение картинки:



И рядом с ней файл с начальными и конечными параметрами:

User parameters:
User name: Andrew
Mode: Terrigen
Well depth(м): 1000
Reservoir permeability (мкм^2): 0.1
Viscosity of the rupture fluid (МПа/с): 0.04
Radius of the well supply circuit (м): 600
Pressure difference (bottom of the well and local mining) (МПа): 2.5
Uncovered layer thickness (м): 13.5
Additional filtration resistance: 1
Result parameters:
New well debit (м^3/сек): 0.000615544
Crack radius (м-12): 0.922966
Crack width (м-3): 207.309
Crack volume (м^3-18): 554.802
Crack length (м): 47.931

Получили результат для нашего месторождения.

5) Выводы

Я реализовал симулятор проведения ГРП с помощью среды разработки Qt Creator, усвоил навыки работы с ООП и встроенные Qt инструменты, понял архитектуру строения программы Qt.