МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №5

По дисциплине ОСиСП за II семестр

«Разработка многопоточных приложений»

Выполнил:
Студент 3-го курса
Группы ПО-5
Игнатюк А.А.
Проверил:
Дряпко А.В.

Брест 2021

Цель работы: Познакомиться с возможностями, предлагаемыми фреймворком Qt, для разработки многопоточных приложений.

Вариант 7: Игра Сапер.

Задание:

1. Основное задание заключается в доработке функционала обновления, разработка которого производилась в ЛР №4. Нужно интегрировать указанную функцию в само приложение, без использования стороннего клиента. При этом серверная часть приложения остается без изменений (возможны некоторые доработки сервера, без изменения общей клиент-серверной архитектуры).
2. Проверка обновления должна осуществляться автоматически по таймеру (QTimer) либо по непосредственному запросу пользователя. Предусмотреть выбор из меню политики обновления (с пользовательским подтверждением, без подтверждения/автоматически).
3. Сам процесс обновления должен осуществляться с использованием отдельного потока (QThread) с минимальной вовлечённостью пользователя.
4. Необходимо отображать прогресс обновления (для этого можно использовать строку состояния - QStatusBar).
5. Для демонстрации процесса обновления и независимой работы основного и вспомогательного потоков приложения осуществлять передачу с сервера обновления, помимо основных обновляемых компонентов (в соответствии с вариантом задания), одного-двух крупных файлов с произвольным содержимым (например, видео).
6. Обновляемые компоненты по вариантам (ЛР №4):
   №7 DLL, конфигурационный файл (количество мин, размер поля)
7. Процесс обновления логируется. При завершении обновления пользователю выдается соответствующее сообщение.

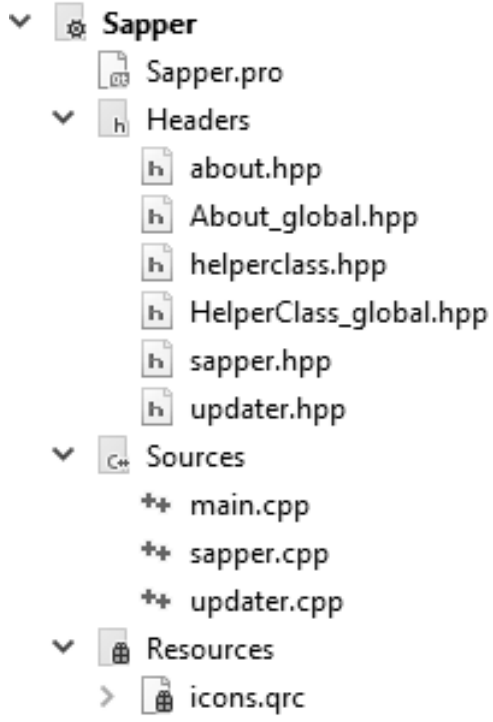

Рисунок 1 - Структура проекта.

Для реализации обновления приложения в отдельном потоке, функционал обновление был вынесен в отдельный класс Updater, который наследуется от класса QThread.

Серверная часть осталась без изменений по сравнению с лабораторной работой №4.

updater.hpp <Select Symbol>

```
#pragma once

#ifndef UPDATER_HPP
#define UPDATER_HPP

#include <QTcpSocket>
#include <QWidget>

class Updater : public QObject {
  Q_OBJECT

 public:
  explicit Updater(QString& v_VersionLink,
                   QObject* const c_QObjectPtr = nullptr);
  ~Updater();

 public slots:
  void sf_process();
  void sf_socket_read();
  void sf_socket_disconnect();

 signals:
  void s_connecting();
  void s_can_not_connect();
  void s_up_to_date();
  void s_updated();
  void s_finished();

  void s_add_progress();
  void s_end_progress();

 private:
  QString& m_VersionLink;
  QTcpSocket* m_SocketPtr{nullptr};

  std::uint64_t m_LogCounter{};
  QString mc_Separator{",,,,,"};
};

#endif  // UPDATER_HPP

```

Рисунок 2 - Содержимое файла updater.hpp.

```
    void sf_connecting();
    void sf_can_not_connect();
    void sf_up_to_date();
    void sf_updated();

    void sf_manual_politic();
    void sf_auto_approval_politic();
    void sf_auto_politic();

    void sf_add_progress();
    void sf_end_progress();
```

Рисунок 3 - Добавление обработчиков новых сигналов в sapper.hpp.

updater.cpp | Updater::sf_socket_read() -> void

```cpp
#include "updater.hpp"

#include <QDir>
#include <QFile>
#include <QMessageBox>

#include <fstream>

Updater::Updater(QString& v_VersionLink, QObject* const c_QObjectPtr)
    : QObject(c_QObjectPtr), m_VersionLink(v_VersionLink) {}

Updater::~Updater() { delete m_SocketPtr; }

void Updater::sf_process() {
  if (m_SocketPtr != nullptr) {
    m_SocketPtr->close();
    delete m_SocketPtr;
  }

  m_SocketPtr = new QTcpSocket(this);
  connect(m_SocketPtr, SIGNAL(readyRead()), this, SLOT(sf_socket_read()));
  connect(m_SocketPtr, SIGNAL(disconnected()), this,
          SLOT(sf_socket_disconnect()));

  m_SocketPtr->connectToHost("127.0.0.1", 2809);
  m_SocketPtr->waitForConnected(1);

  if (m_SocketPtr->state() != QTcpSocket::ConnectedState ||
      !m_SocketPtr->isOpen()) {
    emit s_end_progress();
    emit s_can_not_connect();
    return;
  }

  m_SocketPtr->write(QByteArray::fromStdString(m_VersionLink.toStdString()));
}

void Updater::sf_socket_read() {
  emit s_connecting();

  QByteArray v_RawData{};

  while (m_SocketPtr->bytesAvailable()) {
    emit s_add_progress();
    v_RawData.append(m_SocketPtr->readAll());
    m_SocketPtr->waitForReadyRead(100);
  }

  std::ofstream v_Fout{QString{
      QDir::currentPath() + "/logs/log_" + QString::number(++m_LogCounter) +
      ".txt"}.toStdString()};

  if (v_Fout.is_open()) {
    v_Fout << QString{"Client version: " + m_VersionLink + '\n'}.toStdString();
  } else {
    qDebug() << "Can not save log!";
  }

  std::int64_t v_PositionNow{},
      v_PositionNext{v_RawData.indexOf(",,,,,", v_PositionNow)};

```

Рисунок 4 - Содержимое файла updater.cpp.

Продолжение рисунка 4.

```
  while (v_PositionNow < v_RawData.size()) {
    emit s_add_progress();

    QString v_ResponseCode{};

    if (v_PositionNext == -1) {
      v_ResponseCode =
          v_RawData.sliced(v_PositionNow, v_PositionNow + v_RawData.size());
    } else {
      v_ResponseCode =
          v_RawData.sliced(v_PositionNow, v_PositionNext - v_PositionNow);
    }

    if (v_ResponseCode == "0") {
      emit s_end_progress();
      emit s_up_to_date();
      return;
    }

    v_PositionNow = v_PositionNext + mc_Separator.size();
    v_PositionNext = v_RawData.indexOf(",,,,,", v_PositionNow);

    if (v_ResponseCode == "1") {
      const QString c_Version{

          v_RawData.sliced(v_PositionNow, v_PositionNext - v_PositionNow)};

      v_PositionNow = v_PositionNext + mc_Separator.size();
      v_PositionNext = v_RawData.indexOf(",,,,,", v_PositionNow);

      m_VersionLink = c_Version;
      v_Fout << QString{"Server version: " + m_VersionLink +
                        "\nFiles to receive: "}
                    .toStdString();
      continue;
    }

    if (v_ResponseCode == "2") {
      const QString c_NewFileName{
          v_RawData.sliced(v_PositionNow, v_PositionNext - v_PositionNow)};

      if (v_Fout.is_open()) {
        v_Fout << QString{c_NewFileName + " "}.toStdString();
      }

      v_PositionNow = v_PositionNext + mc_Separator.size();
      v_PositionNext = v_RawData.indexOf(",,,,,", v_PositionNow);

      const QByteArray c_Data{
          v_RawData.sliced(v_PositionNow, v_PositionNext - v_PositionNow)};

      v_PositionNow = v_PositionNext + mc_Separator.size();
      v_PositionNext = v_RawData.indexOf(",,,,,", v_PositionNow);

      QFile v_File(QDir::currentPath() + "/files/" + c_NewFileName);

      if (v_File.open(QIODevice::WriteOnly)) {
        v_File.write(c_Data);
        v_File.close();
      }

      continue;
    }

    break;
  }
```

Продолжение рисунка 4.

```
128
129     if (v_Fout.is_open()) {
130       v_Fout << '\n';
131       v_Fout.close();
132     }
133
134     emit s_end_progress();
135     emit s_updated();
136     emit s_finished();
137   }
138
139   void Updater::sf_socket_disconnect() { m_SocketPtr->close(); }
140
```

```
229 void Sapper::f_load_difficulty() {
230   QFile v_File(QDir::currentPath() + "/files/difficulty.conf");
231
232   if (!v_File.open(QIODevice::ReadOnly | QIODevice::ExistingOnly)) {
233     sf_create_game();
234     return;
235   }
236
237   const QString c_Data{v_File.readLine()};
238   v_File.close();
239
240   if (c_Data == "Easy") {
241     sf_create_easy_game();
242     return;
243   }
244
245   if (c_Data == "Medium") {
246     sf_create_medium_game();
247     return;
248   }
249
250   if (c_Data == "Hard") {
251     sf_create_hard_game();
252     return;
253   }
254
255   if (c_Data.startsWith("New")) {
256     const QStringList c_Values{c_Data.split(' ')};
257     m_BombsCount.insert("New", c_Values[1].toULongLong());
258     m_NewGameRowsCount = c_Values[2].toULongLong();
259
260     m_GameDifficultyMenuPtr->addAction("New", this, SLOT(sf_create_new_game()));
261
262     sf_create_new_game();
263     return;
264   }
265
266   sf_create_game();
267 }
268
```

Рисунок 5 - Изменение функциональности загрузки уровня сложности под возможность использования пользовательских настроек.

```
132     void sf_create_new_game();

438 ▾ void Sapper::sf_create_new_game() {
439     m_CheckedFieldsCount = m_GameDifficulty = m_BombsCount["New"];
440     m_RowsCount = m_NewGameRowsCount;
441     m_GameButtonsCount = m_RowsCount * m_RowsCount;
442     sf_create_game();
443   }
444
```

Рисунок 6 - Определение нового обработчика для создания пользовательского уровня.

```
567 ▾ void Sapper::sf_check_for_updates() {
568 ▾   if (m_UpdaterThreadPtr) {
569       m_UpdaterThreadPtr->exit();
570     }
571
572     m_UpdaterThreadPtr = new QThread{};
573     Updater* v_UpdaterPtr{new Updater{m_Version}};
574
575     v_UpdaterPtr->moveToThread(m_UpdaterThreadPtr);
576
577     connect(m_UpdaterThreadPtr, SIGNAL(started()), v_UpdaterPtr,
578             SLOT(sf_process()));
579
580     connect(v_UpdaterPtr, SIGNAL(s_connecting()), this, SLOT(sf_connecting()));
581     connect(v_UpdaterPtr, SIGNAL(s_can_not_connect()), this,
582             SLOT(sf_can_not_connect()));
583     connect(v_UpdaterPtr, SIGNAL(s_up_to_date()), this, SLOT(sf_up_to_date()));
584     connect(v_UpdaterPtr, SIGNAL(s_updated()), this, SLOT(sf_updated()));
585
586     connect(v_UpdaterPtr, SIGNAL(s_add_progress()), this,
587             SLOT(sf_add_progress()));
588     connect(v_UpdaterPtr, SIGNAL(s_end_progress()), this,
589             SLOT(sf_end_progress()));
590
591     connect(v_UpdaterPtr, SIGNAL(s_finished()), m_UpdaterThreadPtr, SLOT(quit()));
592     connect(v_UpdaterPtr, SIGNAL(s_finished()), v_UpdaterPtr,
593             SLOT(deleteLater()));
594     connect(m_UpdaterThreadPtr, SIGNAL(finished()), m_UpdaterThreadPtr,
595             SLOT(deleteLater()));
596
597 ▾   if (m_UpdateProgressPtr != nullptr) {
598       delete m_UpdateProgressPtr;
599       m_UpdateProgressPtr = nullptr;
600     }
601
602     m_UpdateProgressPtr = new QProgressDialog{this};
603     m_UpdateProgressPtr->setLabelText("Update progress");
604     m_UpdateProgressPtr->setRange(0, 100);
605     m_UpdateProgressPtr->setValue(10);
606     m_UpdateProgressPtr->setAutoClose(false);
607     m_UpdateProgressPtr->setAutoReset(false);
608     m_UpdateProgressPtr->setCancelButtonText("OK");
609     m_UpdateProgressPtr->show();
610
611     m_UpdaterThreadPtr->start();
612   }
613
```

Рисунок 7 - Модификация обработчика обновления с использованием нового класса Updater.

```
614 void Sapper::sf_connecting() {
615   QMessageBox::information(this, "Update Checker",
616                            "Getting data from the server...");
617 }
618
619 void Sapper::sf_can_not_connect() {
620   QMessageBox::information(this, "Update Checker",
621                            "Can not connect to server!");
622 }
623
624 void Sapper::sf_up_to_date() {
625   QMessageBox::information(this, "Update Checker", "You are up to date!");
626 }
627
628 void Sapper::sf_updated() {
629   QMessageBox::information(
630       this, "Update Checker",
631       "Updated successfully!\nPlease restart the program!");
632 }
633
```

Рисунок 8 - Функциональность новых обработчиков в классе Sapper.

```
634 void Sapper::sf_manual_politic() {
635   m_UpdatePolitic = "Manual";
636   m_GameUpdatePoliticsMenuPtr->actions().at(0)->setChecked(true);
637
638   m_UpdateTimerPtr->stop();
639   delete m_UpdateTimerPtr;
640   m_UpdateTimerPtr = nullptr;
641  }
642
643 void Sapper::sf_auto_approval_politic() {
644   m_UpdatePolitic = "Auto approval";
645   m_GameUpdatePoliticsMenuPtr->actions().at(1)->setChecked(true);
646
647   if (m_UpdateTimerPtr != nullptr) {
648     m_UpdateTimerPtr->stop();
649     delete m_UpdateTimerPtr;
650     m_UpdateTimerPtr = nullptr;
651   }
652
653   m_UpdateTimerPtr = new QTimer(this);
654   connect(m_UpdateTimerPtr, &QTimer::timeout, [this]() {
655
656     bool v_Answer = QMessageBox::warning(this, "Update Checker",
657                                          "Want to start the update now?", "No",
658                                          "Yes", QString(), 1, 1);
659
660     if (v_Answer) {
661       sf_check_for_updates();
662     }
663
664     if (m_UpdateTimerPtr != nullptr) {
665       m_UpdateTimerPtr->start(10'000);
666     }
667   });
668
669   m_UpdateTimerPtr->start(10'000);
670  }
671
672 void Sapper::sf_auto_politic() {
673   m_UpdatePolitic = "Auto";
674   m_GameUpdatePoliticsMenuPtr->actions().at(2)->setChecked(true);
675
676   if (m_UpdateTimerPtr != nullptr) {
677     m_UpdateTimerPtr->stop();
678     delete m_UpdateTimerPtr;
679     m_UpdateTimerPtr = nullptr;
680   }
681
```

Рисунок 9 - Функциональность новых политик обновления Sapper.

```
694 void Sapper::sf_add_progress() {
695   if (m_UpdateProgressPtr == nullptr) {
696     return;
697   }
698
699   constexpr int c_INC{10}, c_MAX{90};
700
701   if (m_UpdateProgressPtr->value() < c_MAX) {
702     m_UpdateProgressPtr->setValue(m_UpdateProgressPtr->value() + c_INC);
703   }
704 }
705
706 void Sapper::sf_end_progress() {
707   if (m_UpdateProgressPtr == nullptr) {
708     return;
709   }
710
711   constexpr int c_MAX{100};
712   m_UpdateProgressPtr->setValue(c_MAX);
713 }
714
```

Рисунок 10 - Функциональность обработчиков изменения прогресса обновления.

Вся функциональность по передаче файлов по сети совпадает с прошлой работой, можно получить те же результаты, поэтому подробное тестирование передачи файлов здесь не производилось.

Сервер может отправить пользователю настройки для дополнительного уровня сложности в определенном формате. В связи с этим поле может принимать разные размеры с разным количеством бомб, а не только стандартные как раньше.

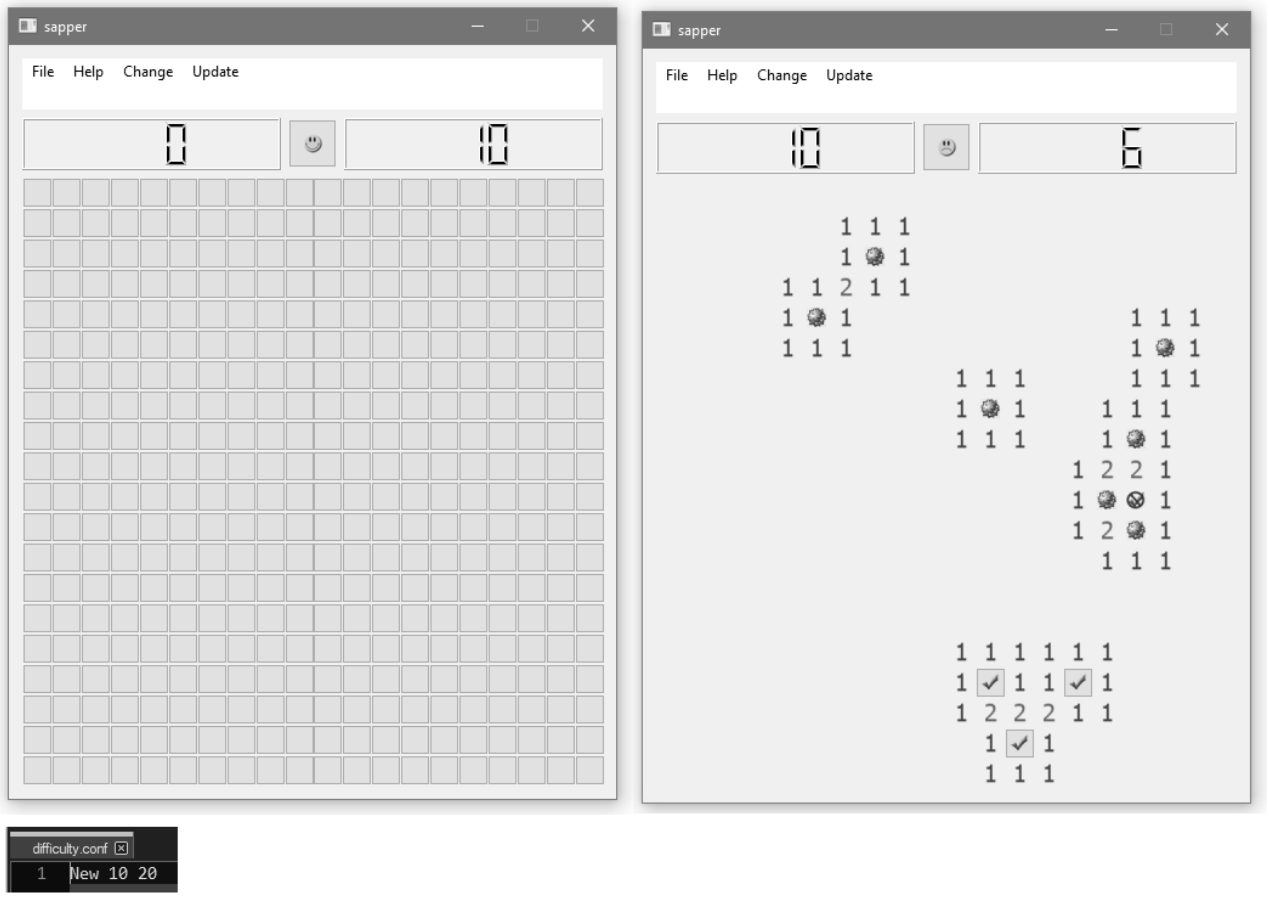

Рисунок 11 - Пример запуска программы с пользовательскими параметрами (10 мин, поле 20 на 20).

Тестирование:

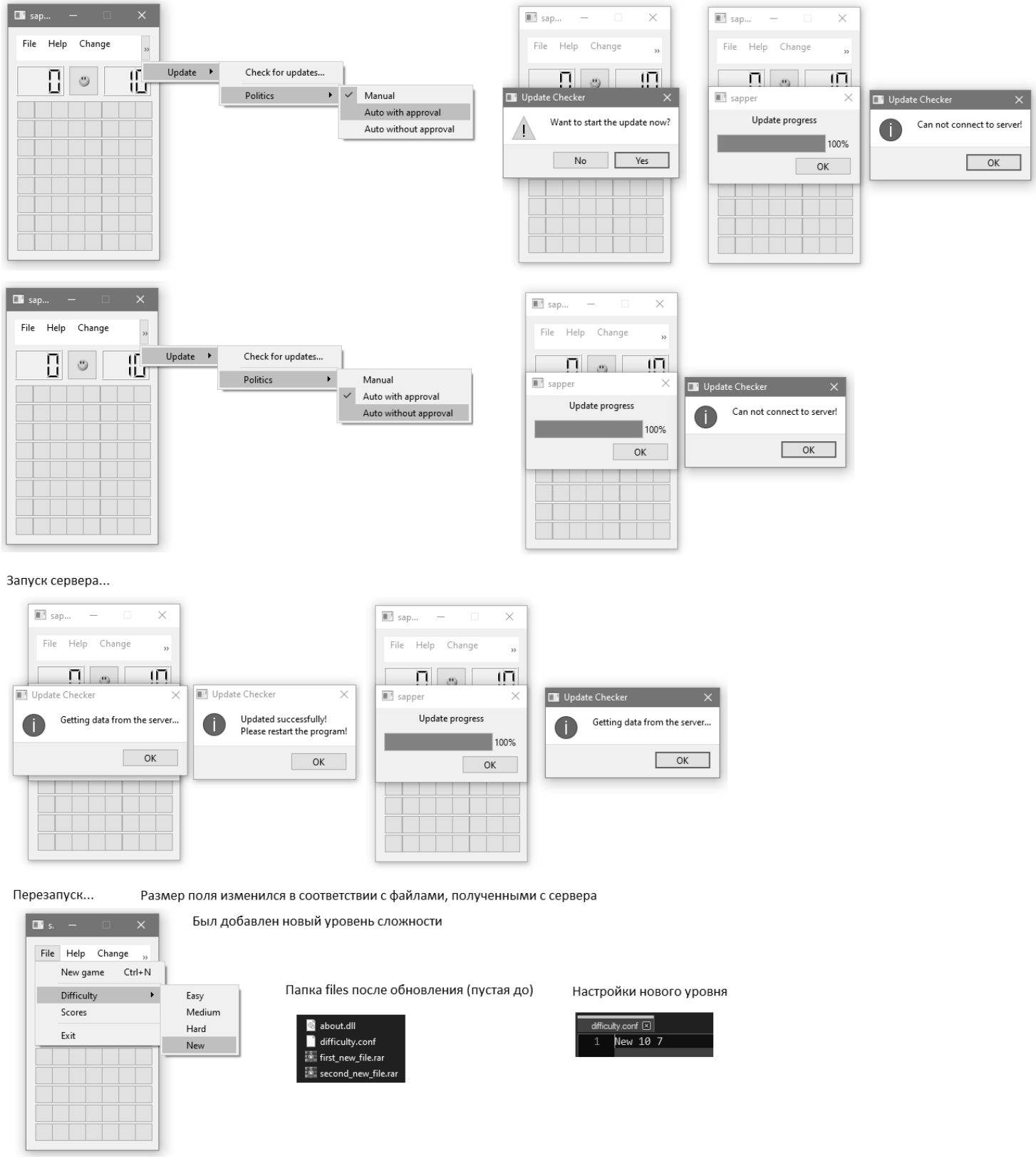

Рисунок 12 – Результат работы программы.

Вывод: Познакомился с возможностями, предлагаемыми фреймворком Qt, для разработки многопоточных приложений.