РАЗРАБОТКА ГРАФИЧЕСКОГО КРОССПЛАТФОРМЕННОГО ПРИЛОЖЕНИЯ «UNIGAME»

Т. Д. Лихоманов, Н. Б. Буторина
Томский государственный университет
lihomanov.t@mail.ru, nnatta07@mail.ru

Введение

Целью работы было реализовать графическое приложение, которое в последствии можно будет назвать полноценной игрой.
Перед нами стояли задачи:

1. Создать максимально интересное приложение для пользователя, способное «тренировать мозг».
2. Реализовать наиболее понятный и приятный глазу пользователя интерфейс.

В написании нашей работы мы вдохновились графическим приложением для платформы Android – «Пифагория».

1. Варианты построения приложений

В настоящее время перед разработчиками программных приложений зачастую стоит вопрос выбора способа реализации приложения. Перед каждым разработчиком стоит выбор: разрабатывать приложение только для одной платформы или делать его кроссплатформенным? У каждого варианта есть свои плюсы и минусы. Каждый вариант по-разному сказывается на ситуациях разработки приложения. Рассмотрение подходов кроссплатформенных приложений стоит рассматривать непосредственно к конкретным условиям разработки этих приложений. Всегда существуют нюансы в выборе способа разработки приложения. Например, приложение, написанное непосредственно только для Windows будет сложно адаптировать под другие операционные системы. Сложно, но не невозможно.
Как правило, разработку любого приложения компания начинает с аналитики рынка, то есть в каких условиях и на каких операционных системах будет востребовано приложение.

2. Нативный подход

Нативный подход – это разработка приложений под конкретную аппаратно-программную платформу. В основном эти приложения пишутся на языках, специально созданных для данной платформы. Нативными приложениями называют те, с которыми пользователь сталкивается с первых дней использования устройства. Например, нативными приложениями можно назвать установленные по умолчанию браузер, будильник, календарь и т.д. на мобильной платформе Android. [2]

3. Кроссплатформенный подход

Кроссплатформенный подход предполагает унификацию большей части программного кода приложения, его независимость от последующего выбора платформы. Как правило, под кроссплатформенной разработкой подразумевают использование специальных фреймворков для написания и создания приложений на основе языка C++. Один из таких фреймворков на данный момент является QT. Мета-объектная система – это часть ядра QT фреймворка для поддержки QT расширений в C++. Инструмент разработки MOC считается частью библиотеки QT. Задача этого инструмента – это поддержка расширений языка C++, необходимые для интроспекции и рефлексии в QT (сюда относятся сигналы, слоты и QML). Необходимость использования MOC является одним из главных объектов критики QT. [2]

4. Фреймворк QT (инструментарий)

Фреймворк – это программная платформа, которая определяет структуру программной системы. Облегчает разработку и объединение разных компонентов большого программного проекта.

QT– это библиотека для создания кроссплатформенных оконных приложений на языке программирования C++. QT стоит рассматривать как полноценный инструмент классов на все случаи жизни. В QT существует возможность разработки приложений не только на языке C++, но и на языке QML, который в свою очередь очень сильно похож на JavaScript. QT направлен на быстрое прототипирование и разработку кроссплатформенных приложений. [1]

Положительные моменты фреймворка:

- В QT существует множество хороших инструментов, которые помогают в разработке приложений. Например, IDE QT Creator, QT Designer, QT Linguist и QT Assistant.
- QT содержит библиотеки, в которых содержится интуитивно понятные API интерфейсы для элементов, такие как анимация, работа с сетями и т.д.

Недостаток:

- В QT очень сложен входной порог начинающим. Чтобы с уверенностью использовать фреймворк QT нужно знать и понимать ключевой механизм взаимодействия объектов – сигналы и слоты.

5. Кроссплатформенная система автоматизации сборки CMake

В нашей работе мы пользовались специальной кроссплатформенной системой автоматизации сборки программного обеспечения из исходного кода – CMake. При этом стоит отметить, что CMake сборкой непосредственно не занимается, а лишь генерирует файлы управления сборкой из файлов CMakeLists.txt.
Для того чтобы собрать проект с помощью CMake, нам необходимо было создать и разместить в корне проекта (где лежит исходный код) файл CMakeLists.txt, который хранит в себе правила и цели сборки.

```
cmake_minimum_required(VERSION 3.15)

project(uniGame VERSION 1.0.0 LANGUAGES CXX)

set(CMAKE_CXX_STANDART 17)

set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)
set(CMAKE_INCLUDE_CURRENT_DIR ON)
```

Здесь мы произвели инициализацию проекта. В начале мы указываем минимальную версию CMake, ниже которой собрать проект не удастся. Следующей строкой прописывается имя проекта и его версию. Стоит отметить, что сигнатура *LANGUAGES* необязательна и указывает какие языки программирования необходимы для построения проекта. В нашем случае поддерживаемыми языками программирования будут: C и C++. Остальным мы говорим CMake для какой версии языка C++ собирается проект.
AUTOMOC - это логическое значение, определяющее, будет ли CMake автоматически обрабатывать препроцессор QT MOC. [3]
AUTORCC - это логическое значение, определяющее, будет ли CMake автоматически обрабатывать генератор кода QT RCC. [3]
Последняя переменная говорит CMake автоматически добавлять в каждый каталог ${CMAKE_CURRENT_SOURCE_DIR} и ${CMAKE_CURRENT_BINARY_DIR} пути включения для этого каталога. Это полезно в основном для сборок вне исходного кода,

где файлы, созданные в дереве сборки, включены в файлы, расположенные в дереве исходного кода. [3]

```
set(CMAKE_PREFIX_PATH "C:\\Qt\\5.13.0\\msvc2017_64")

find_package(Qt5 COMPONENTS Core Widgets Gui Multimedia REQUIRED)

set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)
```

Переменная *CMAKE_PREFIX_PATH* указывает CMake на пакеты QT. Делаем мы это для того, чтобы производить разработку в среде Visual Studio Professional 2019. Так сборщик сможет найти пакеты, чтобы добавить их к проекту. Командой *find_package* мы производим добавление необходимых пакетов к проекту. Опция *REQUIRED* останавливает обработку с сообщением об ошибке, если пакет не может быть найден. И последняя переменная используется для того, чтобы CMake понимал куда поместить все целевые файлы при сборке. [3]

Инициализация переменных, которые указывают на пути к исходным файлам (непосредственно к коду) выглядят так:

```
set(project_sources
        src/main.cpp
        src/Matrix.cpp
        src/GraphicScene.cpp
        src/GraphicView.cpp
)

set(project_ui
        ui/Matrix.ui
)

set(project_resources
        src/resource.qrc
)
```

```
set(project_headers
        include/Configuration.h
        include/GraphicScene.h
        include/GraphicView.h
        include/Matrix.h
        include/Functions.h

        # Triangles levels

        include/levels/ILevel.h
        include/levels/Level_Triangle_1.h
        include/levels/Level_Triangle_1.cpp
        include/levels/Level_Triangle_2.h
        include/levels/Level_Triangle_2.cpp
        include/levels/Level_Triangle_3.h
        include/levels/Level_Triangle_3.cpp

        # Length and distance levels

        include/levels/Level_LengthAndDistance_1.h
        include/levels/Level_LengthAndDistance_1.cpp
        include/levels/Level_LengthAndDistance_2.h
        include/levels/Level_LengthAndDistance_2.cpp
        include/levels/Level_LengthAndDistance_3.h
        include/levels/Level_LengthAndDistance_3.cpp
)
```

Добавление файлов в сам проект записывается следующим образом:

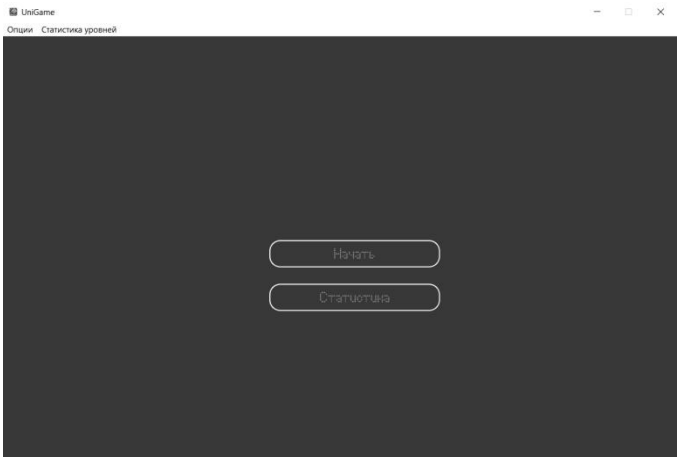
```
add_executable(${PROJECT_NAME} ${GUI_TYPE}
        ${project_headers}
        ${project_sources}
        ${project_headers_wrapped}
)
```

Завершающим моментом для файла CMakeLists.txt является линковка необходимых библиотек к проекту. Благодаря линковке мы производим компоновку, то есть собираем из нескольких подключенных пакетов (объектных модулей) один исполняемый модуль.

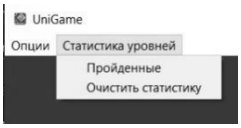
```
target_link_libraries(${PROJECT_NAME}
        PUBLIC
        Qt5::Core
        Qt5::Gui
        Qt5::Widgets
        Qt5::Multimedia
)
```

6. Реализация программы

При начале работы с приложением перед пользователем появляется главное окно.

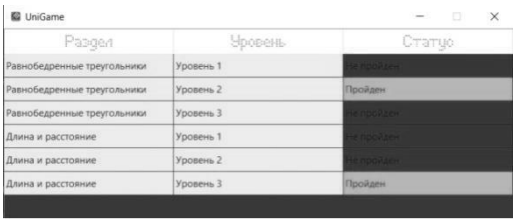


В главном меню находится 2 кнопки, через которые можно управлять приложением.
Кнопка *«Начать»* позволяет пользователю перейти непосредственно в основную игровую область, где находятся игровые уровни, окно отображения и т.д.
Кнопка *«Статистика»* имеет свойство отображения пройденных уровней в отдельном окне, чтобы пользователь мог наблюдать за своими результатами (сколько уровней он прошел, и какие уровни у него ещё не пройдены).
Также в этом окне существует горизонтальное меню, которое находится сверху приложения. В области этого меню находится 2 вкладки, которые в свою очередь являются тоже некими кнопками, через которые производится «общение» пользователя и приложения.
По нажатию на меню «Статистика уровней перед пользователем появляется дополнительное меню с двумя кнопками:

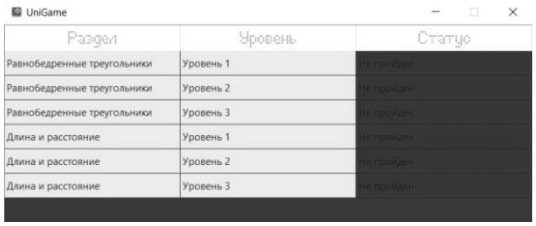


Кнопка *«Пройденные»* открывает перед пользователем окно информации о пройденных и не завершённых уровнях. В этом окне существует три столбца:

1. В первом столбце находится имя раздела.
2. Во втором столбце находится номер уровня.
3. Для третьего столбца устанавливается статус, пройден уровень или нет. Если пользователь успешно прошёл уровень в разделе, то его статус будет выделен зелёным цветом, в случае если наоборот, он ещё не прошёл уровень, то клетка статуса будет выделена красным цветом.



Кнопка *«Очистить статистику»* является функциональной в том смысле, что при нажатии на неё пользователь «говорит» приложению очистить файл статистики, то есть приложение полностью обнулит статистику, и все уровни будут иметь статус «Не пройден».



Для статистики в приложении существует две функции, которые отвечают за инициализацию и сохранение статистики уровней.

```
Q_SLOT void loadStatistic();
Q_SLOT void saveStatistic();
```

Эти две функции являются слотами. В QT слоты используются для получения сигналов, но они также могут быть нормальными функциями-членами. Слот вызывается, когда вырабатывается сигнал, с которым он связан.
Слот *«loadStatistic»* вызывается всегда в конструкторе главного класса *«Matrix»* и имеет следующий алгоритм:

1. В файле ресурсов лежит файл статистики, который необходимо выгрузить в папку с установленной игрой.
2. После успешной выгрузки, файл открывается специальным потоком QT, который называется «QTextStream». Благодаря этому потоку мы можем считывать данные из файла или записывать (изменять) файл статистики.
3. Вся информация об уровнях во время работы приложения будет находиться в оперативной памяти.

Слот *«saveStatistic»* связан со своим сигналом в конструкторе класса *«Matrix»*.

```
connect(this->ui->actionClearStatistics, &QAction::triggered, this, &Matrix::saveStatistic);
```

Слот сохранения будет всегда вызываться при нажатии кнопки очистки статистики в главном меню, и также будет вызываться в деструкторе главного класса при завершении работы приложения.

7. Отрисовывание уровней

Всё отрисовывание уровней производится в центральном окне. При выборе уровня из выпадающего списка QComboBox перед пользователем будет отрисован выбранный уровень. Пользователю даётся право выбрать уровень, который он желает изучить и пройти. При выборе уровня посылается сигнал, который связан со слотом управления и выбора уровней. Для двух разделов сигналы и слоты связаны следующим образом:

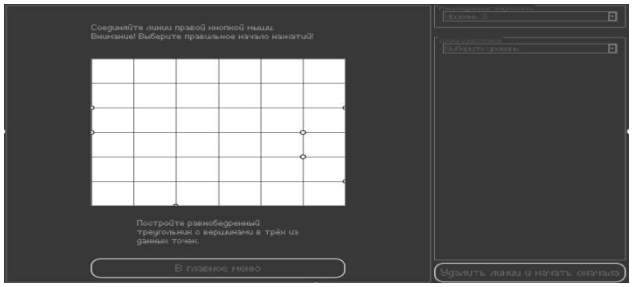
```
// Сигналы и слоты для выбора уровней
connect(this->ui->levelTianglesComboBox, QOverload<int>::of(&QComboBox::currentIndexChanged), this, &Matrix::chooseTriangleLevel);
connect(this->ui->levelLengthAndDistanceComboBox, QOverload<int>::of(&QComboBox::currentIndexChanged), this, &Matrix::chooseLengthAndDistanceLevel);
```

Объект раздела для равнобедренных треугольников связывается с помощью сигнала встроенного в класс QComboBox со слотом *«chooseTriangleLevel»*. Стоит отметить, что соединяемые сигналы и слоты в QT должны иметь идентичные сигнатуры, то есть количество и типы входных аргументов должны совпадать. Исключением является случай, когда сигнал имеет большее число аргументов, чем слот. В этом случае «лишние» аргументы просто не передаются в слот. Но в нашей реализации сигнатуры сигналов и слотов совпадают. Таким образом, при выборе нужного уровня, сигнал передаёт слоту аргумент с номером выбранной строки из выпадающего списка уровней (нумерация строк производится с нуля).

При выборе уровня происходит следующее:

1. Очистка игрового окна.
2. Отрисовка матрицы (сетка в игровом окне).
3. Инициализация нужного уровня и логики.

Также существует дополнительная кнопка в игровом окне, которая отображается не для всех уровней, а только для тех, которым требуется принудительная очистка из-за сложности прохождения уровня. Покажем отрисованное игровое окно для одного уровня:



В нижней части появляется текст с заданием уровня, в верхней части подсказка, чтобы пройти уровень (говорим пользователю какие кнопки мыши ему нужно нажать, чтобы производить отрисовку линий).
Выставляется имя дополнительной кнопки, которая позволяет очистить игровое окно, чтобы пользователь мог начать прохождение уровня сначала. GraphicView связывается со слотами отрисовки линий для текущего уровня, а кнопка очистки со слотом очистки игрового окна.
Уровень начинается со следующей функции:

```
void Level_Triangle_3::startLevel()
{
    // Сразу показываем подсказку / инструкцию к уровню
    this->showHint();

    // Показываем задание
    this->showTooltip();

    // Отрисовываем уровень
    this->paintLevel();
}
```

Самой отрисовкой точек для уровня занимается функция *«paintLevel».* В этой функции существует массив точек, который заполняется так:

```
for (qint32 i = 0; i < 7; ++i)
{
    if (i == 0)
    {
        this->paintPoint(QPoint(0, height * 2));
        points.push_back(QPoint(0, height * 3));
    }

    if (i == 2)
        points.push_back(QPoint(width * i, height * 6));

    if (i == 5)
    {
        this->paintPoint(QPoint(width * i, height * 3));
        points.push_back(QPoint(width * i, height * 4));
    }

    if (i == 6)
    {
        this->paintPoint(QPoint(width * 6, height * 2));
        this->paintPoint(QPoint(width * 6, height * 5));
    }
}
```

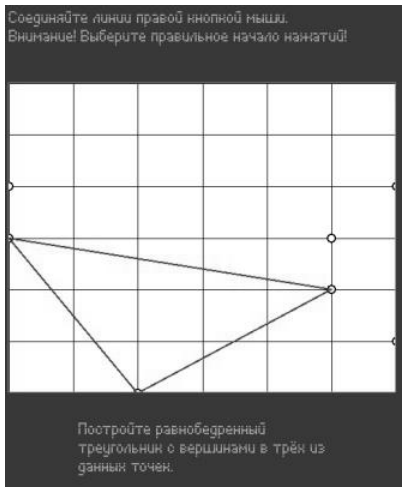
Мы вычисляем нужные места в матрице, где нужно произвести отрисовку точек с помощью функции *«paintPoint».*

Теперь вся «магия» происходит, когда пользователь рисует сами линии, обработка нажатий клавиш мыши происходит в двух слотах:

```
void Level_Triangle_3::paintPointOnGraphicView(QMouseEvent* event)
{
    double rad = 5;
    _scene->addEllipse(QRectF(event->x() - rad, event->y() - rad, rad * 2.0, rad * 2.0), QPen(), QBrush(Qt::yellow));

    _previousPos = event->pos();
}

void Level_Triangle_3::paintLineOnGraphicView(QMouseEvent* event)
{
    if (event->button() == Qt::MouseButton::RightButton)
    {
        if (_previousPos != QPoint())
            _scene->addLine(QLineF(QPointF(_previousPos), QPointF(event->pos())), QPen(QBrush(Qt::yellow), 3));

        _previousPos = event->pos();
    }
}
```

Первый слот добавляет на сцену окружность и сохраняет место нажатия. Второй слот занимается непосредственно отрисовкой линии на сцене.
Отрисовывается это следующим образом:

Как можно видеть, соединены именно «нужные» три точки, которые образуют равнобедренный треугольник. Для проверки правильно ли решён уровень, пользователь должен нажать клавишу «Space» на клавиатуре. Для выбранного уровня вызывается функция проверки, которая проходится по всем точкам и проверяет, связывают ли линии нужные точки на сцене, если все нужные точки связаны правильно, то функция вызывает функцию завершения уровня. Функция завершения «красиво» отрисовывает уровень, соединяя все нужные точки. Также она изменяет статус уровня в статистике на «Пройден».



Заключение

В ходе выполнения нашей работы был изучен кросплатформенный способ разработки приложения на языке C++ с использованием библиотеки QT.
В процессе отладки приложения возникали сложности при изменении файла ресурса статистики. Для их устранения пришлось создать копии файла статистики в корне установленного приложения.
Также при написании приложения пришлось столкнуться с проблемой реализации отрисовки 2D объектов, а именно при связывании объектов и событий нажатием клавиш. Для решения проблемы достаточно было просто переопределить свои классы отрисовки с создания сцен для уровней. В дальнейшем планируется дополнять приложение всевозможными уровнями.

Литература

1. QT. [Электронный ресурс]: Официальная документация QT – 2020. – URL: https://doc.qt.io/
2. Artjoker. [Электронный ресурс]: Нативная или кроссплатформенная разработка? – 2020. – URL: https://artjoker.ua/ru/blog/nativnaya-ili-kross-platformennaya-razrabotka/.
3. CMake. [Электронный ресур]: Документация CMake – 2020. – URL: https://cmake.org/documentation/.