Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

«Национальный исследовательский университет

“Высшая школа экономики”»

###### Факультет информатики, математики и компьютерных наук

###### Базовая кафедра группы компаний МЕРА

Долинин Михаил Андреевич

**Разработка программы имитатора игры в прямоугольный бильярд с графическим интерфейсом**

Курсовая работа студента 2-ого курса бакалавриата группы 13-ПИ по программированию

Научный руководитель

Егоров Е.Е.

Доцент, к.э.н.

Нижний Новгород, 2015

Оглавление

[Введение 3](#_Toc420015259)

[Описание теоретической части работы 4](#_Toc420015260)

[Общие правила игры в русский бильярд 4](#_Toc420015261)

[Инструменты разработки 6](#_Toc420015262)

[Описание практической части приложения 9](#_Toc420015263)

[Заключение 10](#_Toc420015264)

[Использованная литература 12](#_Toc420015265)

[Приложения 13](#_Toc420015266)

## **Введение**

**Бильярд** — собирательное название нескольких настольных игр с разными правилами, а также специальный стол, на котором происходит игра. Это игра для двух и более игроков специальными шарами на специальном столе, покрытом сукном, при этом шары перемещаются по столу с помощью ударов кием по ним. Цель игры — с помощью ударов кием по особому шару, битку, загнать все остальные шары в лузы, специальные отверстия, расположенные по периметру стола, раньше соперника. Существуют различные варианты игры, отличающиеся размером и формой стола и начальным расположением шаров на нем, правилами бильярда и другим.

В процессе данной работы было создано приложение, позволяющее играть в бильярд двум игрокам.

**Цель работы:** Разработка программы имитатора игры в прямоугольный бильярд с графическим интерфейсом.

**Задачи:**

1. Изучение правил игры в бильярд
2. Изучение механики и разработка математической модели для реализации игры в бильярд
3. Изучение библиотеки QT
4. Изучение вспомогательное библиотеки qwt
5. Изучение библиотеки OpenGL

**Актуальность:** Актуальность приложения считаю значительной в виду того, что редко встречается программа игры в бильярд, написанная с использованием библиотек Qt, qwt и OpenGL, что представляется странным, так как указанные выше библиотеки имеют огромные функционал и располагают полным набором необходимых средств для реализации приложений данного вида. Кроме того, работа с этими библиотеками удобна и не представляет огромной сложности при достаточном изучении благодаря подробному help'у по каждой функции библиотеки. Также следует отметить, что библиотека Qt, которая является кроссплатформенной, в настоящее время активно развивается и набирает все большую популярность, несмотря на свою долгую жизнь в мире IT-разработки. Она с успехом используется во многих современных приложениях, таких как Skype, Oovoo и другие.

Для реализации был выбран русский бильярд, как наиболее популярный в нашей стране вид бильярда.

## **Описание теоретической части работы**

### **Общие правила игры в русский бильярд**

Партия начинается с расстановки шаров. Набор из пятнадцати белых шаров, пронумерованных от 1-15, выставляются на бильярдный стол. С помощью треугольника они ставятся вплотную друг к другу, располагаясь таким образом, чтоб передний шар был на отметке, расположенной в центре задней половине игрового поля стола, а основание пирамиды параллельно короткому борту.

Любая игра в бильярд начинается с удара, не удивительно, потому что, он является одним из основных составляющих игры. Соответственно существует несколько видов удара и связанные с ним действия и классифицируются с точки зрения на правильный или неправильный удар.  
Началом удара считается касание набойки кия к битку. Соударение – т.е. сильный толчок, кия по битку и как результат следующее за ним движения шара или шаров – начало игры. Касание битка с прицельным шаром является обязательным условием любого правильного удара. Окончанием удара следует считать остановку всех шаров, по всей поверхности стола, включая шары, вращающиеся на одном месте. Удар кием по битку (шару по которому наносят удар кием) выполняется набойкой кия в направлении его продольной оси. При этом должно соблюдаться обязательное условие, при котором одна нога игрока должна стоять на полу или касаться его. При нарушении этого правила игрок штрафуется. В момент удара все шары должны быть неподвижными. Выскочившие, неправильно забитые шары, должны быть выставленными в произвольном порядке на линии выставления шаров от задней отметки в направлении заднего борта.

Видов ударов в бильярде так же несколько. Они делятся на простые и сложные:   
Простые - это клапштос, накат, оттяжка, правый и левый боковик.  
"Клапштос" — основной удар, удар в центр битка. Этот удар основывается на законе физики: шар, движущийся по плоскости, ударившись о другой, равный ему не только по массе и объему, но и по упругости, неизбежно останавливается на месте, в точке соприкосновения, а свою скорость движения и его направление передает другому, тому, о какой он ударился.  
"Накат" — удар в верхнюю точку строго над центром битка он считается одним из самых легких в игре.  
"Оттяжка" — удар в нижнюю точку строго под центром битка. При этом виде удара действуют 2 силы: это сила удара, заставляющая шар продвигаться вперед и сила обратного вращения, действующая оттого, что удар нанесен в точку шара, расположенную ниже центра.  
"Правый боковик" — удар в среднюю точку на правой стороне битка.  
"Левый боковик" — удар в среднюю точку на левой стороне битка.

Сложными ударами считаются:  
"Верхний правый боковик" — удар в верхнюю точку на правой стороне битка.  
"Верхний левый боковик" — удар в верхнюю точку на левой стороне битка.  
"Нижний правый боковик" — удар в нижнюю точку на правой стороне битка.  
"Нижний левый боковик" — удар в нижнюю точку на левой стороне битка.

Второй главной составляющей в бильярде является положение шара. Положение шара определяется положением его центра относительно разметки игрового стола. Шар, стоящий в точности на линии дома (прямая, проведенная через переднюю отметку параллельно переднему борту.), считается расположенным вне него, относительно выполняющего удар игрока.

Шар считается правильно забитым, т.е. сыгранным, если он упал в лузу в результате правильного удара. Любой удар (за исключением начального) считается правильным, если не нарушено ни одно из положений Правил, предъявляемым Федерацией Русского бильярда.

### **Инструменты разработки**

Qt **—** кроссплатформенный инструментарий разработки ПО на языке программирования C++. Есть также «привязки» ко многим другим языкам программирования: Python — PyQt, PySide; Ruby — QtRuby[8]; Java — Qt Jambi[9]; PHP — PHP-Qt и другие.

Qt позволяет запускать написанное с его помощью ПО в большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Включает в себя все основныеклассы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементовграфического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования.

Отличительная особенность Qt от других библиотек — использование *Meta Object Compiler (MOC)* — предварительной системы обработки исходного кода (в общем-то, Qt — это библиотека не для чистого C++, а для его особого наречия, с которого и «переводит» MOC для последующей компиляции любым стандартным C++ компилятором). MOC позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как *слоты* и *сигналы*.

**Сигналы и слоты** — это подход, который позволяет реализовать шаблон «наблюдатель», минимизируя написание повторяющегося кода. Концепция заключается в том, что компонент (часто виджет) может посылать сигналы, содержащие информацию о событии (например: был выделен текст «слово», была открыта вторая вкладка). В свою очередь другие компоненты могут принимать эти сигналы посредством специальных функций — слотов. Система сигналов и слотов хорошо подходит для описания Графического интерфейса пользователя, именно поэтому библиотека Qt была выбрана мной для написания курсовой работы. Также механизм сигналов/слотов может быть применён для асинхронного ввода-вывода (включая сокеты, pipe, устройства с последовательным интерфейсом, др.) или уведомления о событиях. В библиотеке Qt благодаря Метаобъектному компилятору отпадает необходимость писать код регистрации/дерегистрации/вызова, так как эти шаблонные участки кода генерируются автоматически.

Утилита MOC ищет в заголовочных файлах на C++описания классов, содержащие макрос Q\_OBJECT, и создаёт дополнительный исходный файл на C++, содержащий *метаобъектный* код.

Qt позволяет создавать собственные плагины и размещать их непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Qt комплектуется визуальной средой разработки графического интерфейса «Qt Designer», позволяющей создавать диалоги и формы в режиме WYSIWYG. В поставке Qt есть «Qt Linguist» — графическая утилита, позволяющая упростить локализацию и перевод программы на многие языки; и «Qt Assistant» — справочная система Qt, упрощающая работу с документацией по библиотеке, а также позволяющая создавать кросс-платформенную справку для разрабатываемого на основе Qt ПО. Начиная с версии 4.5.0 в комплект Qt включена среда разработки «Qt Creator», которая включает в себя редактор кода, справку, графические средства «Qt Designer» и возможность отладки приложений. «Qt Creator» может использовать GCC или Microsoft VC++ в качестве компилятора и GDB в качестве отладчика. Для Windows версий библиотека комплектуется компилятором, заголовочными и объектными файлами MinGW.

#### Qwt

**Qwt** или **Qt Widgets for Technical Applications** — набор Qt-виджетов и вспомогательных классов, необходимых для создания графического представления числовых данных. Помимо виджета для двумерного отображения данных (QwtPlot) он включает в себя классы для отображения данных в разных масштабах осей, различные стили отображения кривых и маркеров на виджете QwtPlot, а также некоторые другие вспомогательные виджеты.

#### OpenGL

**OpenGL** (Open Graphics Library) — спецификация, определяющая независимый от языка программированияплатформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях. На платформеWindows конкурирует с Direct3D.

OpenGL ориентируется на следующие две задачи:

* Скрыть сложности адаптации различных 3D-ускорителей, предоставляя разработчику единый API.
* Скрыть различия в возможностях аппаратных платформ, требуя реализации недостающей функциональности с помощью программной эмуляции.

Основным принципом работы OpenGL является получение наборов векторных графических примитивов в виде точек, линий и многоугольников с последующей математической обработкой полученных данных и построением растровой картинки на экране и/или в памяти. Векторные трансформации и растеризация выполняются графическим конвейером (graphics pipeline), который по сути представляет собой дискретный автомат. Абсолютное большинство команд OpenGL попадают в одну из двух групп: либо они добавляют графические примитивы на вход в конвейер, либо конфигурируют конвейер на различное исполнение трансформаций.

OpenGL является низкоуровневым процедурным API, что вынуждает программиста диктовать точную последовательность шагов, чтобы построить результирующую растровую графику (императивный подход). Это является основным отличием от дескрипторных подходов, когда вся сцена передается в виде структуры данных (чаще всего дерева), которое обрабатывается и строится на экране. С одной стороны, императивный подход требует от программиста глубокого знания законов трёхмерной графики и математических моделей, с другой стороны — даёт свободу внедрения различных инноваций.

## **Описание практической части приложения**

В соответствии с ТЗ для разработки приложения были выбраны библиотеки Qt, Qwt, OpenGL как наиболее подходящие для разработки программы с графическим интерфейсом. Также система сигналов и слотов очень удобна для написания игр, что является несомненным плюсом библиотеки Qt.

#### Общие сведения о реализации

В соответствии с идеологией Qt были созданы файлы main.cpp, mainwindow.cpp, myqwtplot.cpp, qbilliard.cpp и соответсвующе им файлы заголовков mainwindow.h, myqwtplot.h, qbilliard.h, где описаны прототипы функций, реализующих приложение.

В файле mainwindow.cpp содержится описание класса MainWindow, унаследованного от стандартного класса QMainWindow. В нем описывается инициализация стола для игры в бильярд, положение луз, функция обновления изображения и четыре слота: слот, описывающий удар по битку кием, слот, обрабатывающий нажатие мыши, и еще два слова: начинающий игру(делает все элементы интерфейса видимыми) и завершающий ее.

В файле myqwtplot.cpp содержащий реализацию двух функций для обработки мыши, которые являются вспомогательными для реализации удара по битку кием, сигнал удара по битку кием и слот, который также является вспомогательным для реализации удара по битку.

В файле qbilliard.cpp содержится реализация физической модели игры, на которую был сделан упор при разработке данного приложения, так как вся игра в бильярд основана на чистейшей физике. Файл содержит описание положения шаров для игры, функцию движения шаров по столу, функцию удара по битку, функцию взаимодействия между шарами, функцию просчета траекторий движения шаров, реализованную с помощью принципа очереди, благодаря чему траектории движения просчитываются в момент нанесения удара по битку и функцию контроля попаданию шаров в лузы.

Мною были составлены UML-диаграммы классов и последовательностей(Приложения 1,2).

Примеры кода в Приложении 4.

## **Заключение**

Использование совокупности библиотек Qt и Qwt позволяет существенно снизить сложность реализации графических изображений, в частности, игр. Также эти библиотеки обеспечивают лаконичность кода программы, так как система сигналов и слотов позволяет одной строкой заменить существенный объем кода.

Описанный мной выше подход является, на мой взгляд, одним из лучших для создания игр и графических приложений в целом, так как плюсы его очевидны. А поставляемая вместе с библиотекой Qt среда разработки обеспечивает полную, подробную и понятную справку по всему перечню функций, предоставляемых библиотекой. Однако, возможности библиотеки не ограничиваются созданием только лишь графических приложений — Qt нашла широчайшее применение в огромнейшем круге задач, а учитывая ее постоянное развитие — то ли еще будет.

## **Использованная литература**

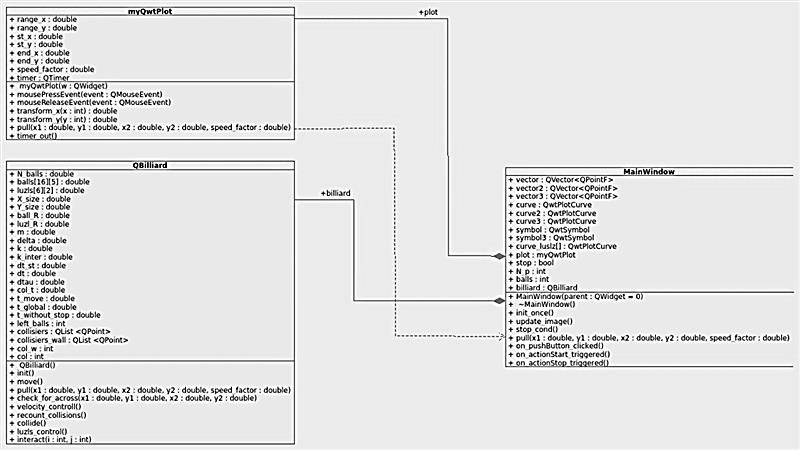
1. Шлее М. Qt 5.3. Профессиональное программирование на C++. — СПб.:

БХВ-Петербург, 2015. — 928 с.:

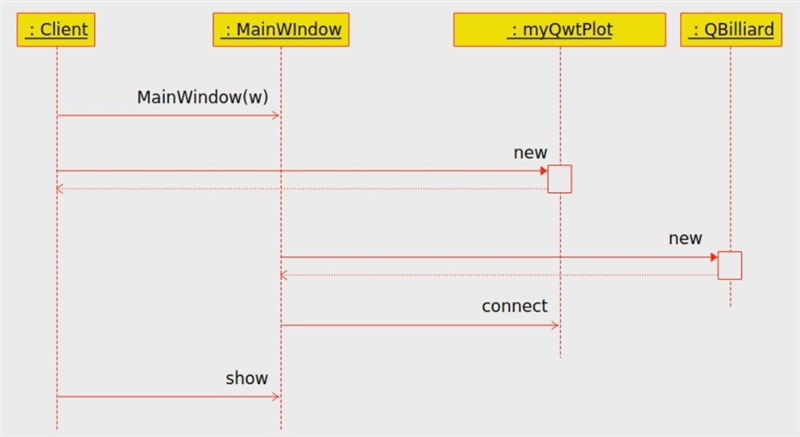
1. Alan Ezust, Paul Ezust. An Introduction to Design Patterns in C++ with Qt 4. 2006
2. habrahabr.ru

## **Приложения**

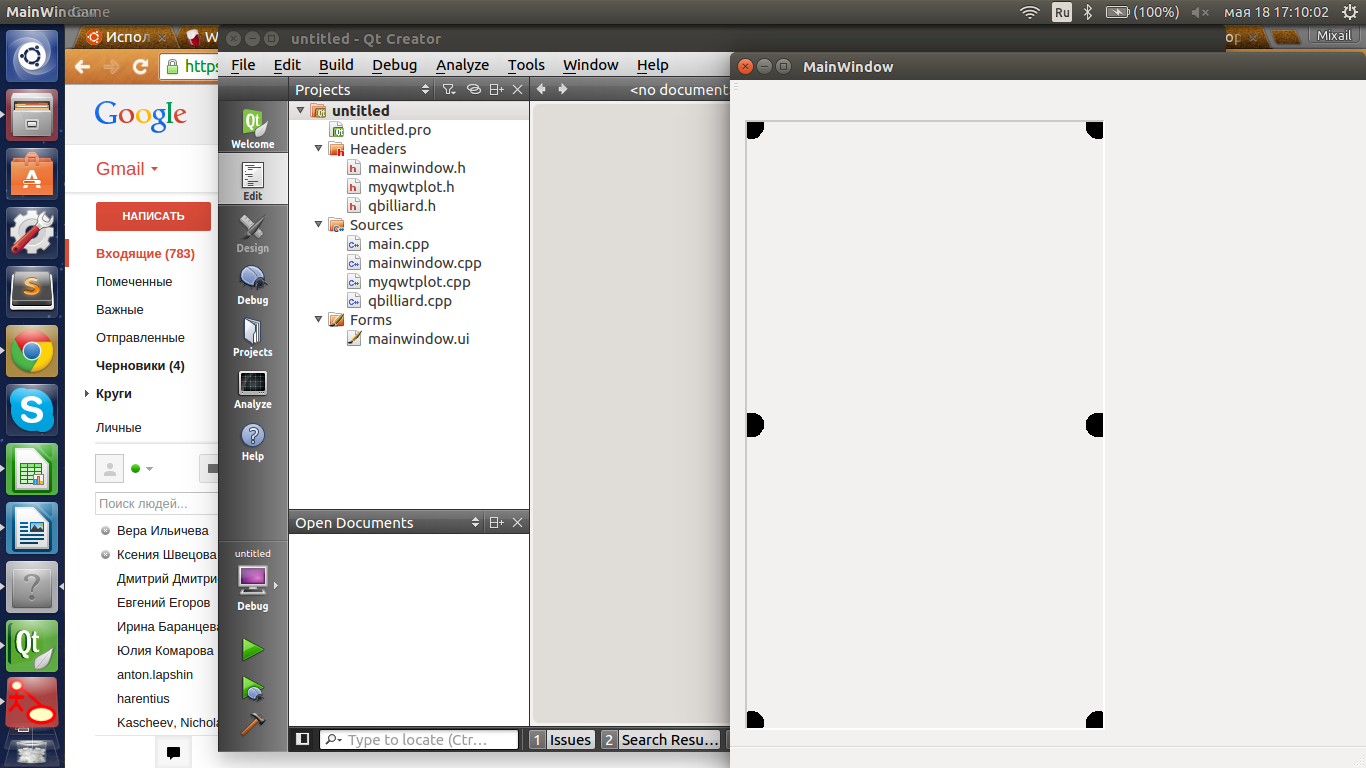
Приложение №1. UML-диаграмма классов.

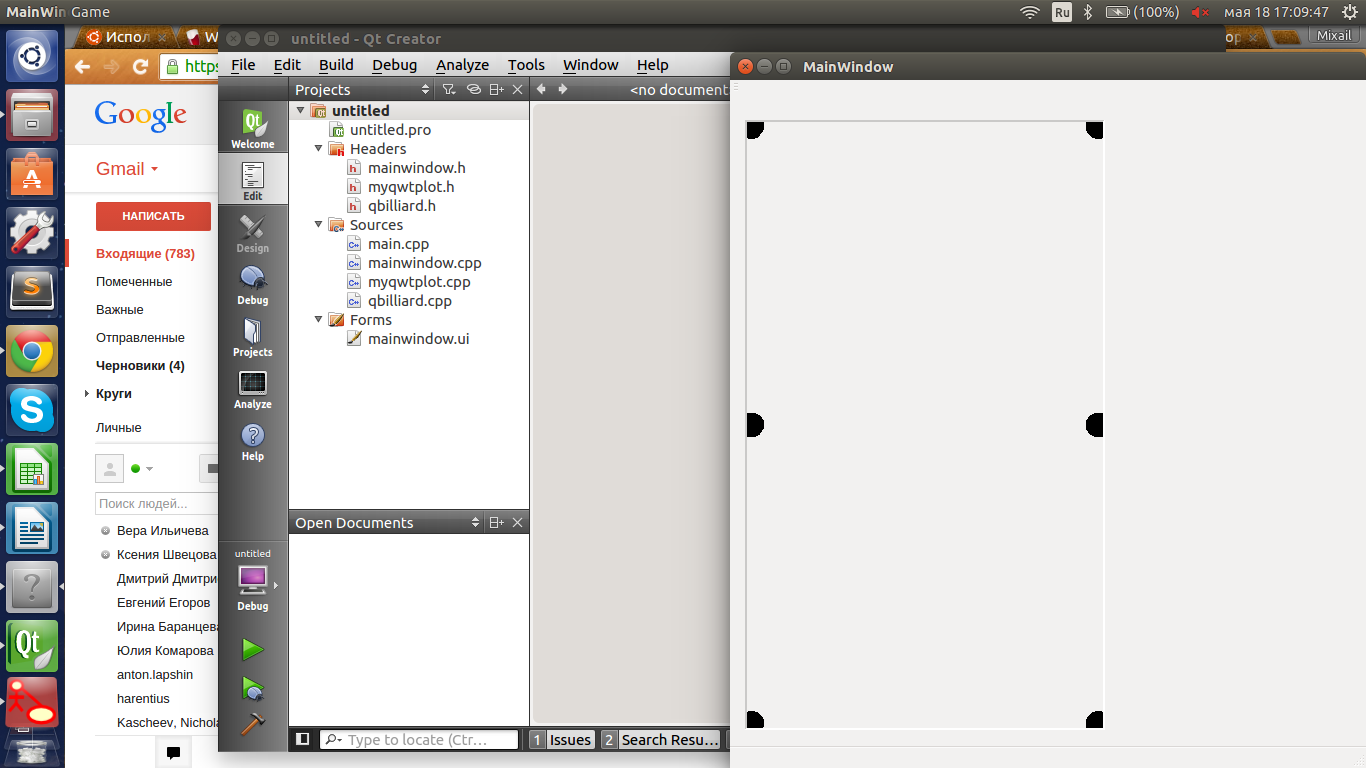
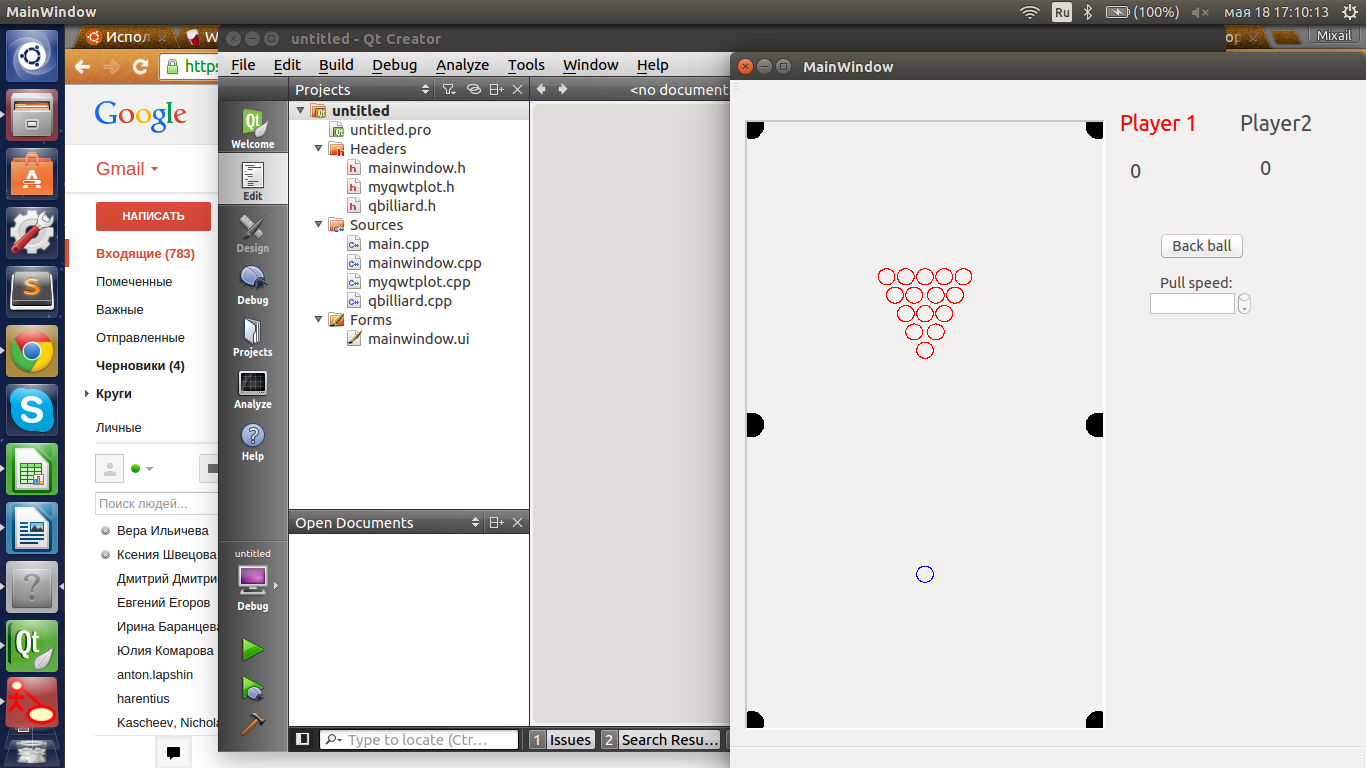


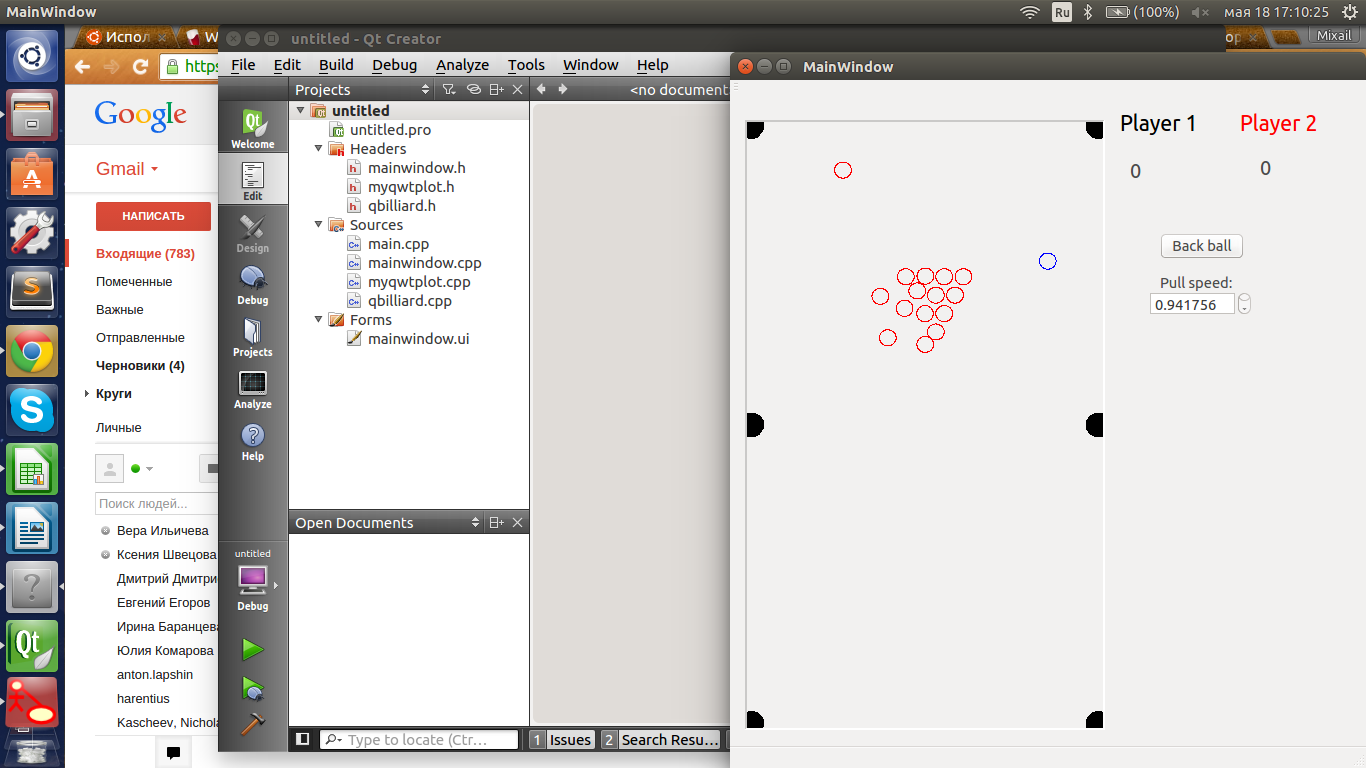
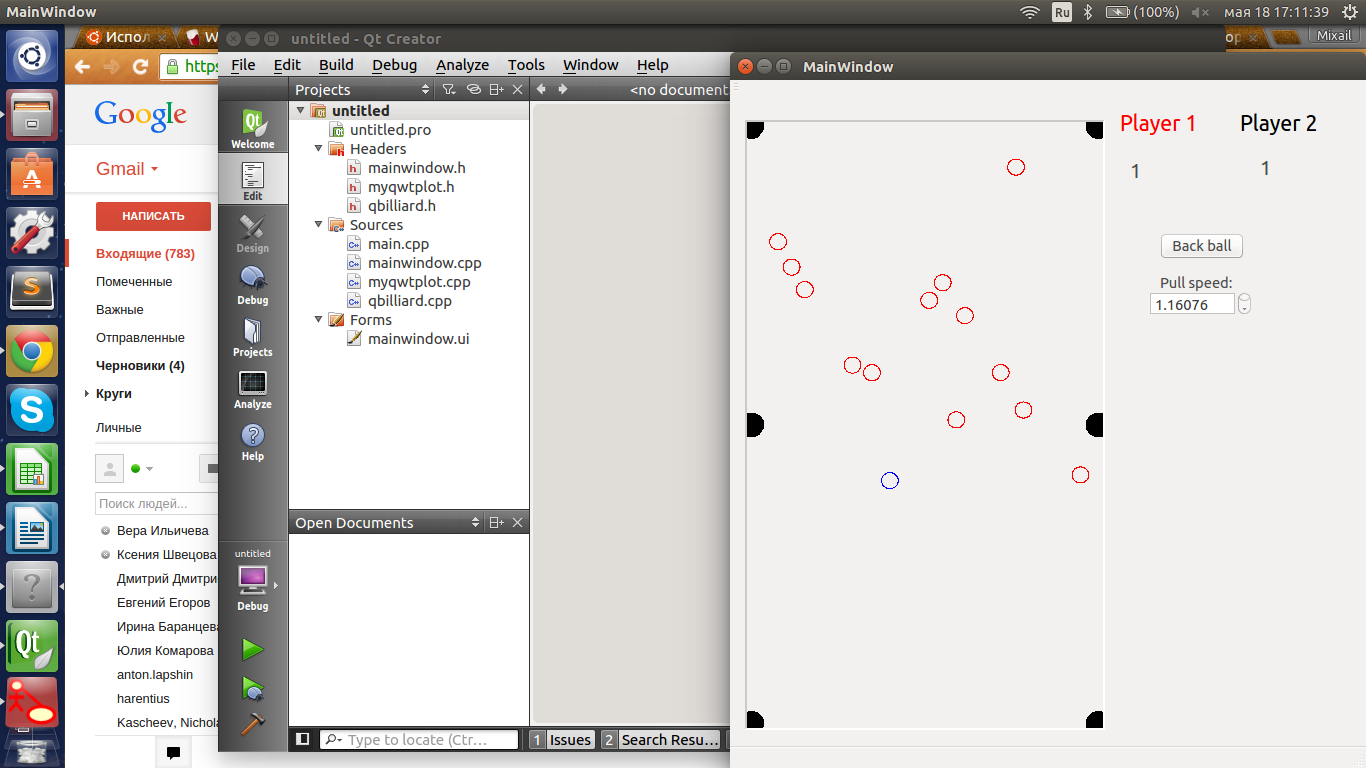
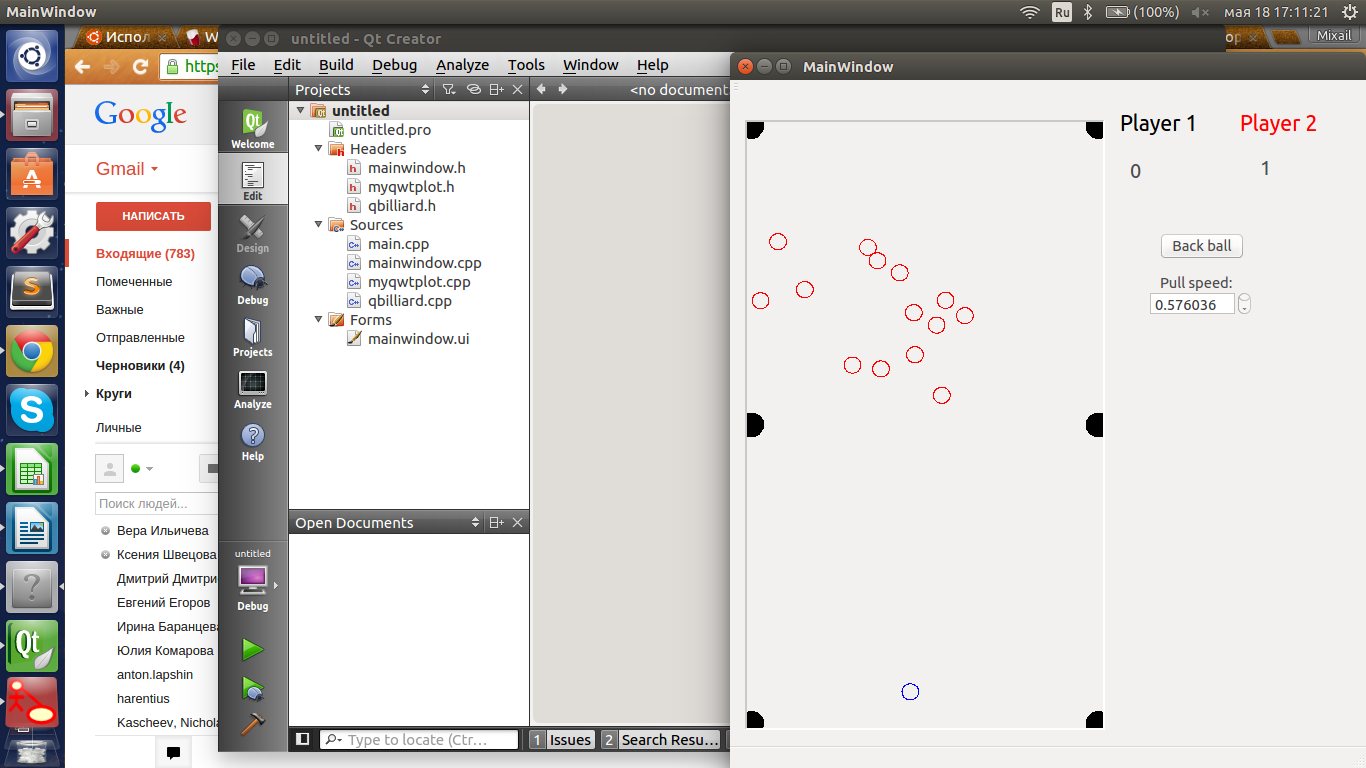
Приложений №2. UML-диаграмма последовательностей.



Приложение №3







Приложение №4. Критические участки кода

Функции, отвечающие за обработку сигналов мыши

void myQwtPlot::*mousePressEvent*(QMouseEvent \*event)

{

st\_x = transform\_x(event->x());

st\_y = transform\_y(event->y());

speed\_factor = 0.0;

timer->start(10);

}

void myQwtPlot::*mouseReleaseEvent*(QMouseEvent \*event)

{

end\_x = transform\_x(event->x());

end\_y = transform\_y(event->y());

timer->stop();

emit pull(st\_x, st\_y, end\_x, end\_y, speed\_factor);

}

double myQwtPlot::transform\_x(int x)

{

return (double)(x - pos().x())\*range\_x/width();

}

double myQwtPlot::transform\_y(int y)

{

return (double)(height() -y -pos().y())\*range\_y/(double)height();

}

Функция, обрабатывающая столкновения шаров

void QBilliard::recount\_collisions()

{

collisiers.clear();

//recounting minimum time

QList<QPoint> mols\_collised\_wall;

QList<QPoint> mols\_collised;

this->t\_without\_stop = 10000;

for (int i = 0; i < 16; i++) {

mols\_collised.clear();

long double tau = 10000;

long double tau\_temp = 10000;

//wals

//x

if (balls[i][4] != 0.0) {

if (balls[i][2] != 0) {

if (balls[i][2] > 0){

tau\_temp = (X\_size - ball\_R - balls[i][0])/balls[i][2];

if (tau\_temp < tau) {

tau = tau\_temp;

mols\_collised\_wall.clear();

mols\_collised\_wall.append(QPoint(i,-2));

}

}

else {

tau\_temp = (ball\_R - balls[i][0])/balls[i][2];

if (tau\_temp < tau) {

tau = tau\_temp;

mols\_collised\_wall.clear();

mols\_collised\_wall.append(QPoint(i,-2));

}

}

}

//y

if (balls[i][3] != 0) {

if (balls[i][3] > 0){

tau\_temp = (Y\_size - ball\_R - balls[i][1])/balls[i][3];

if (tau\_temp < tau) {

tau = tau\_temp;

mols\_collised\_wall.clear();

mols\_collised\_wall.append(QPoint(i,-1));

}

}

else {

tau\_temp = (ball\_R - balls[i][1])/balls[i][3];

if (tau\_temp < tau) {

tau = tau\_temp;

mols\_collised\_wall.clear();

mols\_collised\_wall.append(QPoint(i,-1));

}

}

}

}

//One with other

long double a, b, c, d, dx, dy, dvx, dvy, x1, x2;

for (int j = i + 1; j < 16; j++){

if (balls[j][4] != 0) {

dx = balls[j][0] - balls[i][0];

dy = balls[j][1] - balls[i][1];

dvx = balls[i][2] - balls[j][2];

dvy = balls[i][3] - balls[j][3];

a = qPow(dvx, 2) + qPow(dvy,2);

b = -2\*(dx\*dvx+dy\*dvy);

c = qPow(dx, 2) + qPow(dy, 2) - 4\*ball\_R\*ball\_R;

d = qPow(b,2) - 4\*a\*c;

if ((d >= 0) && ( b<0 )){

d = qPow(d, 0.5);

x1 = (-b - d)/(2\*a);

x2 = (-b + d)/(2\*a);

if ((x1 > 0) && (x2 > 0)){

if (x1 < x2)

tau\_temp = x1;

else tau\_temp = x2;

}

else {

if (x1>0) tau\_temp = x1;

if (x2>0) tau\_temp = x2;

}

if (tau\_temp==tau) {

mols\_collised.append(QPoint(i,j));

}

if (tau\_temp < tau){

mols\_collised.clear();

mols\_collised\_wall.clear();

mols\_collised.append(QPoint(i,j));

tau = tau\_temp;

}

}

}

}

tau = 1.0/k\*qLn(1.0/(1 - k\*tau));

if (tau < dt\_st)

dt = tau/2;

else

dt = dt\_st;

mols\_collised.operator +=(mols\_collised\_wall);

if (tau == (t\_without\_stop - t\_global)){

collisiers.operator +=(mols\_collised);

}

if (tau < (t\_without\_stop - t\_global) && (tau > 0)){

t\_without\_stop = t\_global + tau;

collisiers.clear();

collisiers.operator +=(mols\_collised);

}

}

}