Титульник

Оглавление

[1. Введение 3](#_Toc8742103)

[2. Предметная область 3](#_Toc8742104)

[3. Требование к программе и ПО 4](#_Toc8742105)

[3.1 Техническая часть 4](#_Toc8742106)

[3.2 Требования к интерфейсу программы 5](#_Toc8742107)

[3.3 Требования к составу и параметрам технических средств 5](#_Toc8742108)

[3.4 Требования к программным средствам, используемым программой 6](#_Toc8742109)

[4.Сдача и приемка работы 6](#_Toc8742110)

[5. Структуры данных и алгоритмы 6](#_Toc8742111)

[6. Реализация программы 8](#_Toc8742112)

[7. UML-диаграмма 10](#_Toc8742113)

[8. Заключение 10](#_Toc8742114)

[9. Список литературы 11](#_Toc8742115)

[10. Приложение №1. Исходный код программы 12](#_Toc8742116)

# Введение

Язык C++ является одним из самых популярных языков программирования и имеет широкую область применения. Она вкюлчает в себя создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр).

В данной курсовой работе будет показано одно из применений данного языка, а именно – Разработка игры. Будет рассмотрен пример простейшей баллистической игры.

# Предметная область

Так как разработка баллистической игры проводилась в кроссплатформенной свободной IDE Qt Creator, вся программа основана на уже описанных классах, таких как, например, QApplication (он содержит главный цикл обработки событий, где обрабатываются и координируются все события из оконной системы и других источников). Мы подключаем их с помощью директивы #include. Данная программа также содержит такие классы, как QWidget,QPaintEvent, QPainter, QTimer, QMessageBox, QLCDNumber, QSlider, QVBoxLayout, QPushButton, QHBoxLayout, QGridLayout.

Имеет место и создание собственных классов. Класс CannonField, который наследуется от QWidget, занимается почти всем, что связано с игрой. Именно он рисует поле игры, кнопки, обе баллисты, изменение угла дула каждой из них, выстрел, отслеживает попадание.

В программе присутствуют еще два собственных класса, такие как LCDRange и MyWidget. Оба класса наследуются от QWidget.

Класс LCDRange выполняет создание слайдера и жидкокристаллического индикатора, а также связывает их.

Класс MyWidget содержит конструктор, выполняющий сборку всех элементов и добавление их на сам виджет.

Баллистическая игра реализована в виде приложения с графическим пользовательским интерфейсом с использованием фреймворка Qt.

# 3. Требование к программе и ПО

## 3.1 Техническая часть

В разных концах экрана расположены две баллисты, принадлежащие разным игрокам. Игроки ходят по очереди. Ход заключается в выборе начальной скорости камня и угла между вектором начальной скорости и горизонтом. По этим данным программа должна произвести расчет траектории полета камня и смоделировать полет на экране. После первого игрока ходит второй и т.д. Игра заканчивается, когда один из камней попадет в баллисту противника.

* Программа должна позволять пользователю вводить и изменять угол и начальную скорость.
* Программа должна рассчитывать траекторию выстрела.
* Программа должна заново рассчитывать от изменения входных параметров.
* Программа должна выводить окна сообщений в зависимости от того, кто из двух игроков выиграл.

1. Получение информации о введенном угле и введенной начальной скорости (вводятся с помощью слайдеров)
2. Расчет траектории с помощью стандартной формулы Ньютона для движения в гравитационном поле без учета трения воздуха
3. Предыдущие пункты выполняются для обеих пушек
4. В зависимости от того, какой игрок попал в цель, выводятся окна сообщений, выводящие информацию о том, какой из игроков выиграл.

## 3.2 Требования к интерфейсу программы

При запуске программы должен выводиться виджет на котором должны располагаться:

* Специальное поле, выделенное цветом, отличным от цвета самого виджета. В нижней правой и нижней левой частях поля должны быть расположены две отрисованные баллисты, по умолчанию нацеленные на угол 45 градусов.
* По бокам виджета должны быть расположены в целом 4 слайдера и 4 ЖК-индикатора. Слайдеры, связанные со значением угла, должны быть выставлены на значение 45. Слайдеры, связанные со значением начальных скоростей, должны быть выставлены на значение 10.
* В верхней части виджета должны находиться 3 кнопки: кнопка «Quit» и 2 кнопки «Shoot»

## 3.3 Требования к составу и параметрам технических средств

В состав технических средств должен входить IВМ-совместимый персональный компьютер (ПЭВМ), включающий в себя:   
 1. Процессор Pentium-2.0Hz, не менее;

2. Оперативную память объемом, 1 Гигабайт, не менее;

3. Любую операционную систему;

4. Стандартный пакет С++;

## 3.4 Требования к программным средствам, используемым программой

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией любой операционной системы и ПО C++.

# 4.Сдача и приемка работы

При приемке системы будут проведены следующие испытания:

* Проверка функциональности всех разделов системы.
* Проверка оформления, указанного в Техническом задании.
* Проверка на наличие уязвимостей.
* Передача программы малой группе людей, до этого не знакомых с ней для проверки интуитивно понятного интерфейса.

# Структуры данных и алгоритмы

Начнем с класса CannonField.   
  
 Класс наследуется также от QObject, поэтому используем макрос Q\_OBJECT, чтобы избежать ошибок в работе программы. В public задаем конструктор класса, который рисует «поле битвы», а также задает значения переменных, указанных в private; задаем четыре константных функции, которые возвращают значения четырех private переменных.   
  
 В классе будет два типа слотов. Все слоты в public slots делаем публичными, потому что их используют не только в данном классе, но и в классе MyWidget. setAngle() и setSecAngle() устанавливают значения угла возвышения.setForce() и setSecForce()устанавливают значения начальной скорости.Вызов слотов shoot() и SecShoot() приводит к выстрелу пушки, если снаряд еще не в воздухе.   
  
 Закрытые слоты moveShot(), moveSecShot() используются для перемещения снаряда пока он находится в воздухе, используя QTimer.Они также проверяют, не пересекся ли снаряд с целью.

paintEvent() - виртуальная функция (поэтому указываем protected), которая вызывается Qt всякий раз, когда виджету нужно обновиться (т.е., нарисовать поверхность виджета).

Закрытые функции paintCannon(), paintShot() и paintSecShot() осуществляют отрисовку баллист и снарядов. Функция cannonRect() и SecCannonRect() возвращают ограждающий прямоугольник баллисты в координатах виджета. Сначала мы создаем прямоугольник с размерами 50 x 50, а затем передвигаем его так, чтобы его нижний левый/правый угол совпадал с нижним левым/правым углом виджета. Функции shotRect() и SecShotRect()вычисляют центральную точку снаряда и возвращает ограничивающий прямоугольник для снаряда. Далее в классе объявляются закрытые переменные.

Класс LCDRange.

Как и в классе CannonField, в LCDRange используется макрос Q\_OBJECT. LCDRange() - конструктор класса, value() - открытая функция для получения доступа к значению LCDRange, setValue() - слот, а valueChanged() - сигнал.setRangeдобавляет возможность установки диапазона значений LCDRange.

Также создаем указатель [\*slider](https://vk.com/slider) на объект класса QSlider. Мы можем это сделать, потому что в заголовочном файле lcdrange.h указан class QSlider.

Класс MyWidget содержит только лишь конструктор, который выполняет сборку всех элементов и добавление их на сам виджет.

КлассDistAng включает в себя характеристики полета: расчёт времени, траектории, при чем с учетом гравитации.

# Реализация программы

При запуске программы появится окно (рис. 1).

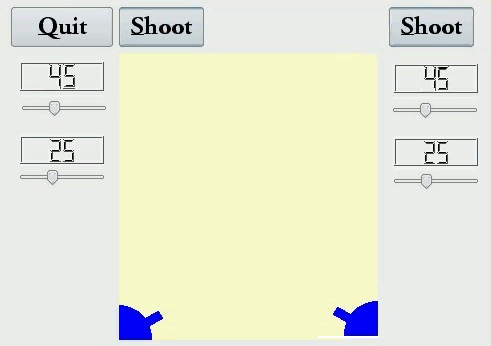


Рис.1

Пользователи сами выбирают за какую из баллист будут играть и кто начинает первым. Пользователи могут как угодно изменять значения слайдеров (рис. 2).

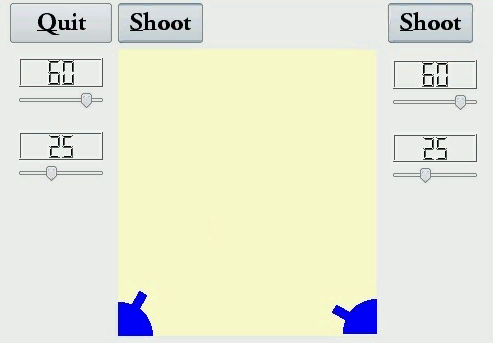


Рис.2

Выстрел производится при нажатии кнопок «Shoot», каждая относится к своей пушке. Реализация выстрела показана на рисунке 3.

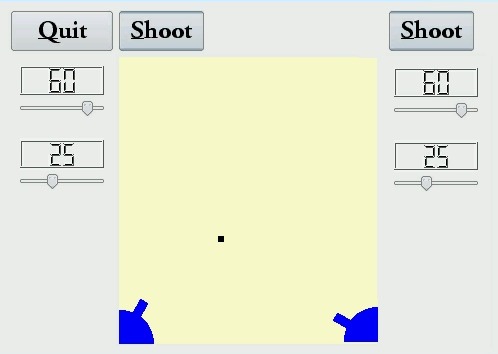
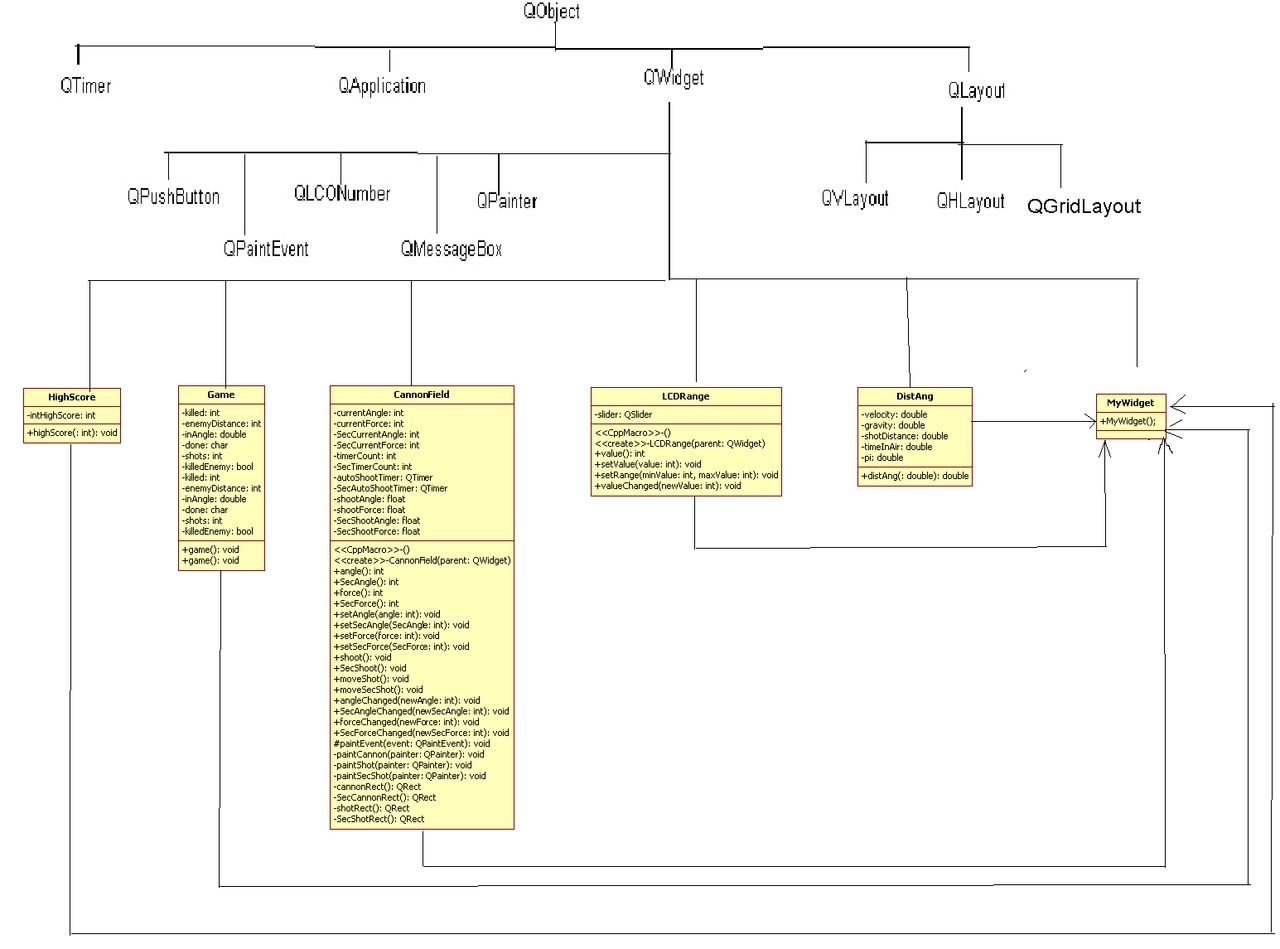


Рис. 3

Выход из игры реализуется с помощью кнопки «Выход».

# UML-диаграмма



# Заключение

В данной курсовой работе была разработана и реализована простейшая баллистическая игра.

При работе над проектом было составлено Техническое задание, по которому создавалась система. А так же были изучены элементы создания графического интерфейса с помощью классов Qt.

Применение объектно-ориентированного подхода к программированию и широкие возможности языка С++ позволяют создавать программы, имеющие мощную функциональность, а использование графического интерфейса позволяет стороннему пользователю без труда использовать готовую систему.

# Список литературы

1. А. Александреску. Современное программирование на C++. Обобщенное программирование и прикладные шаблоны проектирования. Книга для опытных программистов не С++. 2002 год, 330 стр.
2. Т. А. Павловская. C/C++. Программирование на языке высокого уровня. Из серии "300 лучших учебников".2003 год. 461 стр.
3. Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник. 2005 год. 1100 стр.
4. Бланшет, Саммерфилд - Qt4 Программирование GUI на С++. 2ed. 2008
5. Шлее Макс - Профессиональное программирование на C++. +CD. Qt 4.8. 2012
6. Марк Саммерфилд - Qt Профессиональное программирование (High tech). 2011

# Приложение №1. Исходный код программы

## ****main.cpp****

|  |
| --- |
| #include <QApplication> |
|  | #include <QPushButton> |
|  | #include <QHBoxLayout> |
|  | #include <QVBoxLayout> |
|  | #include <QGridLayout> |
|  |  |
|  | #include "cannonfield.h" |
|  | #include "lcdrange.h" |
|  |  |
|  | class MyWidget : public QWidget |
|  | { |
|  | public: |
|  | MyWidget(QWidget \*parent = 0); |
|  | }; |
|  |  |
|  | MyWidget::MyWidget(QWidget \*parent) |
|  | : QWidget(parent) |
|  | { |
|  | QPushButton \*quit = new QPushButton("Выйти"); |
|  | connect(quit, SIGNAL(clicked()), qApp, SLOT(quit())); |
|  |  |
|  | //угол первой пушки |
|  | LCDRange \*angle = new LCDRange; |
|  | angle->setRange(5, 70); |
|  |  |
|  | //угол второй пушки |
|  | LCDRange \*SecAngle = new LCDRange; |
|  | SecAngle->setRange(5,70); |
|  |  |
|  | //сила первой пушки |
|  | LCDRange \*force = new LCDRange; |
|  | force->setRange(10, 50); |
|  |  |
|  | //сила второй пушки |
|  | LCDRange \*SecForce = new LCDRange; |
|  | SecForce->setRange(10, 50); |
|  |  |
|  | CannonField \*cannonField = new CannonField; |
|  |  |
|  |  |
|  | //1 |
|  | connect(angle, SIGNAL(valueChanged(int)), |
|  | cannonField, SLOT(setAngle(int))); |
|  |  |
|  | connect(cannonField, SIGNAL(angleChanged(int)), |
|  | angle, SLOT(setValue(int))); |
|  |  |
|  | //2 |
|  | connect(SecAngle, SIGNAL(valueChanged(int)), |
|  | cannonField, SLOT(setSecAngle(int))); |
|  |  |
|  | connect(cannonField, SIGNAL(SecAngleChanged(int)), |
|  | SecAngle, SLOT(setValue(int))); |
|  |  |
|  | //1 |
|  | connect(force, SIGNAL(valueChanged(int)), |
|  | cannonField, SLOT(setForce(int))); |
|  |  |
|  | connect(cannonField, SIGNAL(forceChanged(int)), |
|  | force, SLOT(setValue(int))); |
|  |  |
|  | //2 |
|  | connect(SecForce, SIGNAL(valueChanged(int)), |
|  | cannonField, SLOT(setSecForce(int))); |
|  |  |
|  | connect(cannonField, SIGNAL(SecForceChanged(int)), |
|  | SecForce, SLOT(setValue(int))); |
|  |  |
|  | QPushButton \*shoot = new QPushButton("Стреляй"); |
|  | connect(shoot, SIGNAL(clicked()),cannonField, SLOT(shoot())); |
|  |  |
|  | QPushButton \*SecShoot = new QPushButton("Стреляй"); |
|  | connect(SecShoot, SIGNAL(clicked()),cannonField, SLOT(SecShoot())); |
|  |  |
|  | QHBoxLayout \*topLayout = new QHBoxLayout; |
|  | topLayout->addWidget(shoot); |
|  | topLayout->addWidget(SecShoot); |
|  |  |
|  | QVBoxLayout \*leftLayout = new QVBoxLayout; |
|  | leftLayout->addWidget(angle); |
|  | leftLayout->addWidget(force); |
|  |  |
|  | QVBoxLayout \*rightLayout = new QVBoxLayout; |
|  | rightLayout->addWidget(SecAngle); |
|  | rightLayout->addWidget(SecForce); |
|  |  |
|  | QGridLayout \*gridLayout = new QGridLayout; |
|  | gridLayout->addWidget(quit, 0, 0); |
|  | gridLayout->addLayout(topLayout, 0, 1); |
|  | gridLayout->addLayout(leftLayout, 1, 0); |
|  | gridLayout->addLayout(rightLayout,1,2); |
|  | gridLayout->addWidget(cannonField, 1, 1, 2, 1); |
|  | gridLayout->setColumnStretch(1, 10); |
|  | setLayout(gridLayout); |
|  |  |
|  | angle->setValue(45); |
|  | force->setValue(10); |
|  | SecForce->setValue(10); |
|  | SecAngle->setValue(45); |
|  | angle->setFocus(); |
|  | SecAngle->setFocus(); |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | QApplication app(argc, argv); |
|  | MyWidget widget; |
|  | widget.setGeometry(100, 100, 500, 355); |
|  | widget.setFixedSize(620,355); |
|  | widget.show(); |
|  | return app.exec(); |
|  | } |

## ****cannonfield.cpp****

|  |
| --- |
|  |
| #include <QPaintEvent> |
|  | #include <QPainter> |
|  | #include <QTimer> |
|  |  |
|  | #include "cannonfield.h" |
|  | #include <math.h> |
|  |  |
|  | CannonField::CannonField(QWidget \*parent) |
|  | : QWidget(parent) |
|  | { |
|  | currentAngle = 45; |
|  | currentForce = 10; |
|  | SecCurrentAngle = 45; |
|  | SecCurrentForce = 10; |
|  | timerCount = 0; |
|  | autoShootTimer = new QTimer(this); |
|  | connect(autoShootTimer, SIGNAL(timeout()), this, SLOT(moveShot())); |
|  | shootAngle = 0; |
|  | shootForce = 0; |
|  | SecTimerCount = 0; |
|  | SecAutoShootTimer = new QTimer(this); |
|  | connect(SecAutoShootTimer,SIGNAL(timeout()),this,SLOT(moveSecShot())); |
|  | SecShootAngle = 0; |
|  | SecShootForce = 0; |
|  | setPalette(QPalette(QColor(250, 250, 200))); |
|  | setAutoFillBackground(true); |
|  | } |
|  |  |
|  | void CannonField::setAngle(int angle) |
|  | { |
|  | if (angle < 5) |
|  | angle = 5; |
|  | if (angle > 70) |
|  | angle = 70; |
|  | if (currentAngle == angle) |
|  | return; |
|  | currentAngle = angle; |
|  | update(cannonRect()); |
|  | emit angleChanged(currentAngle); |
|  | } |
|  |  |
|  | void CannonField::setSecAngle(int SecAngle) |
|  | { |
|  | if (SecAngle < 5) |
|  | SecAngle = 5; |
|  | if (SecAngle > 70) |
|  | SecAngle = 70; |
|  | if (SecCurrentAngle == SecAngle) |
|  | return; |
|  | SecCurrentAngle = SecAngle; |
|  | update(SecCannonRect()); |
|  | emit SecAngleChanged(SecCurrentAngle); |
|  | } |
|  |  |
|  | void CannonField::setForce(int force) |
|  | { |
|  | if (force < 0) |
|  | force = 0; |
|  | if (currentForce == force) |
|  | return; |
|  | currentForce = force; |
|  | emit forceChanged(currentForce); |
|  | } |
|  |  |
|  | void CannonField::setSecForce(int SecForce) |
|  | { |
|  | if (SecForce < 0) |
|  | SecForce = 0; |
|  | if (SecCurrentForce == SecForce) |
|  | return; |
|  | SecCurrentForce = SecForce; |
|  | emit SecForceChanged(SecCurrentForce); |
|  | } |
|  |  |
|  | void CannonField::shoot() |
|  | { |
|  | if (autoShootTimer->isActive()) |
|  | return; |
|  | timerCount = 0; |
|  | shootAngle = currentAngle; |
|  | shootForce = currentForce; |
|  | autoShootTimer->start(5); |
|  | } |
|  |  |
|  | void CannonField::SecShoot() |
|  | { |
|  | if (SecAutoShootTimer->isActive()) |
|  | return; |
|  | SecTimerCount = 0; |
|  | SecShootAngle = SecCurrentAngle; |
|  | SecShootForce = SecCurrentForce; |
|  | SecAutoShootTimer->start(5); |
|  | } |
|  |  |
|  | void CannonField::moveShot() |
|  | { |
|  | QRegion region = shotRect(); |
|  | ++timerCount; |
|  |  |
|  | QRect shotR = shotRect(); |
|  |  |
|  | if (shotR.x() > width() || shotR.y() > height()) { |
|  | autoShootTimer->stop(); |
|  | } else { |
|  | region = region.united(shotR); |
|  | } |
|  | update(region); |
|  | } |
|  |  |
|  | void CannonField::moveSecShot() |
|  | { |
|  | QRegion SecRegion = SecShotRect(); |
|  | ++SecTimerCount; |
|  |  |
|  | QRect SecShotR = SecShotRect(); |
|  |  |
|  | if (SecShotR.x() > width() || SecShotR.y() > height()) { |
|  | SecAutoShootTimer->stop(); |
|  | } else { |
|  | SecRegion = SecRegion.united(SecShotR); |
|  | } |
|  | update(SecRegion); |
|  | } |
|  |  |
|  | void CannonField::paintEvent(QPaintEvent \*) |
|  | { |
|  | QPainter painter(this); |
|  |  |
|  | paintCannon(painter); |
|  | if (autoShootTimer->isActive()) |
|  | paintShot(painter); |
|  | if (SecAutoShootTimer->isActive()) |
|  | paintSecShot(painter); |
|  | } |
|  |  |
|  | void CannonField::paintShot(QPainter &painter) |
|  | { |
|  | painter.setPen(Qt::NoPen); |
|  | painter.setBrush(Qt::black); |
|  | painter.drawRect(shotRect()); |
|  | } |
|  |  |
|  | void CannonField::paintSecShot(QPainter &painter) |
|  | { |
|  | painter.setPen(Qt::NoPen); |
|  | painter.setBrush(Qt::black); |
|  | painter.drawRect(SecShotRect()); |
|  | } |
|  |  |
|  | const QRect barrelRect(30, -5, 20, 10); |
|  | const QRect SecBarrelRect(-50,-5,20,10); |
|  |  |
|  | void CannonField::paintCannon(QPainter &painter) |
|  | { |
|  | painter.setPen(Qt::NoPen); |
|  | painter.setBrush(Qt::black); |
|  |  |
|  | painter.save(); |
|  | painter.translate(0, height()); |
|  | painter.drawPie(QRect(-35, -35, 70, 70), 0, 90 \* 16); |
|  | painter.drawPie(QRect(width()-35,-35,70,70),1450, 90 \* 16); |
|  | painter.rotate(-currentAngle); |
|  | painter.drawRect(barrelRect); |
|  | painter.rotate(currentAngle); |
|  | painter.translate(width(),0); |
|  | painter.rotate(SecCurrentAngle); |
|  | painter.drawRect(SecBarrelRect); |
|  | painter.restore(); |
|  | } |
|  |  |
|  | QRect CannonField::cannonRect() const |
|  | { |
|  | QRect result(0, 0, 50, 50); |
|  | result.moveBottomLeft(rect().bottomLeft()); |
|  | return result; |
|  | } |
|  |  |
|  | QRect CannonField::SecCannonRect() const |
|  | { |
|  | QRect SecResult(0, 0, 50, 50); |
|  | SecResult.moveBottomRight(rect().bottomRight()); |
|  | return SecResult; |
|  | } |
|  |  |
|  | QRect CannonField::shotRect() const |
|  | { |
|  | const double gravity = 4; |
|  |  |
|  | double time = timerCount / 20.0; |
|  | double velocity = shootForce; |
|  | double radians = shootAngle \* 3.14159265 / 180; |
|  |  |
|  | double velx = velocity \* cos(radians); |
|  | double vely = velocity \* sin(radians); |
|  | double x0 = (barrelRect.right() + 5) \* cos(radians); |
|  | double y0 = (barrelRect.right() + 5) \* sin(radians); |
|  | double x = x0 + velx \* time; |
|  | double y = y0 + vely \* time - 0.5 \* gravity \* time \* time; |
|  |  |
|  | QRect result(0, 0, 6, 6); |
|  | result.moveCenter(QPoint(qRound(x), height() - 1 - qRound(y))); |
|  | return result; |
|  | } |
|  |  |
|  | QRect CannonField::SecShotRect() const |
|  | { |
|  | const double SecGravity = 4; |
|  | const double pi = 3.14159265; |
|  |  |
|  | double SecTime = SecTimerCount / 20.0; |
|  | double SecVelocity = SecShootForce; |
|  | double SecRadians = (SecShootAngle \* pi) / 180; |
|  |  |
|  | double SecVelx = (SecVelocity \* cos(pi-SecRadians)); |
|  | double SecVely = SecVelocity \* sin(pi-SecRadians); |
|  | double SecX0 = (SecBarrelRect.left() - 5) \* cos(pi-SecRadians); |
|  | double SecY0 = (SecBarrelRect.left() + 5) \* sin(pi-SecRadians); |
|  | double SecX = SecX0 + SecVelx \* SecTime; |
|  | double SecY = SecY0 + SecVely \* SecTime - 0.5 \* SecGravity \* SecTime \* SecTime; |
|  |  |
|  | QRect SecResult(0, 0, 6, 6); |
|  | SecResult.moveCenter(QPoint(width() + qRound(SecX) , height() - 1 - qRound(SecY))); |
|  | return SecResult; |
|  | } |
|  |  |

## ****lcdrange.cpp****

|  |
| --- |
| #include <QLCDNumber> |
|  | #include <QSlider> |
|  | #include <QVBoxLayout> |
|  |  |
|  | #include "lcdrange.h" |
|  |  |
|  | LCDRange::LCDRange(QWidget \*parent) |
|  | : QWidget(parent) |
|  | { |
|  | QLCDNumber \*lcd = new QLCDNumber(2); |
|  | lcd->setSegmentStyle(QLCDNumber::Filled); |
|  |  |
|  | slider = new QSlider(Qt::Horizontal); |
|  | slider->setRange(0, 99); |
|  | slider->setValue(0); |
|  |  |
|  | connect(slider, SIGNAL(valueChanged(int)), |
|  | lcd, SLOT(display(int))); |
|  | connect(slider, SIGNAL(valueChanged(int)), |
|  | this, SIGNAL(valueChanged(int))); |
|  |  |
|  |  |
|  | QVBoxLayout \*layout = new QVBoxLayout; |
|  | layout->addWidget(lcd); |
|  | layout->addWidget(slider); |
|  | setLayout(layout); |
|  |  |
|  | setFocusProxy(slider); |
|  | } |
|  |  |
|  | int LCDRange::value() const |
|  | { |
|  | return slider->value(); |
|  | } |
|  |  |
|  | void LCDRange::setValue(int value) |
|  | { |
|  | slider->setValue(value); |
|  | } |
|  |  |
|  | void LCDRange::setRange(int minValue, int maxValue) |
|  | { |
|  | if (minValue < 0 || maxValue > 99 || minValue > maxValue) { |
|  | qWarning("LCDRange::setRange(%d, %d)\n" |
|  | "\tRange must be 0..99\n" |
|  | "\tand minValue must not be greater than maxValue", |
|  | minValue, maxValue); |
|  | return; |
|  | } |
|  | slider->setRange(minValue, maxValue); |
|  | } |

## ****cannonfield.h****

|  |
| --- |
| #ifndef CANNONFIELD\_H |
|  | #define CANNONFIELD\_H |
|  |  |
|  | #include <QWidget> |
|  |  |
|  | class CannonField : public QWidget |
|  | { |
|  | Q\_OBJECT |
|  |  |
|  | public: |
|  | CannonField(QWidget \*parent = 0); |
|  |  |
|  | int angle() const { return currentAngle; } |
|  | int SecAngle() const { return SecCurrentAngle; } |
|  | int force() const { return currentForce; } |
|  | int SecForce() const {return SecCurrentForce;} |
|  |  |
|  | public slots: |
|  | void setAngle(int angle); |
|  | void setSecAngle(int SecAngle); |
|  | void setForce(int force); |
|  | void setSecForce(int SecForce); |
|  | void shoot(); |
|  | void SecShoot(); |
|  |  |
|  | private slots: |
|  | void moveShot(); |
|  | void moveSecShot(); |
|  |  |
|  | signals: |
|  | void angleChanged(int newAngle); |
|  | void SecAngleChanged(int newSecAngle); |
|  | void forceChanged(int newForce); |
|  | void SecForceChanged(int newSecForce); |
|  |  |
|  | protected: |
|  | void paintEvent(QPaintEvent \*event); |
|  |  |
|  | private: |
|  | void paintCannon(QPainter &painter); |
|  | void paintShot(QPainter &painter); |
|  | void paintSecShot(QPainter &painter); |
|  | QRect cannonRect() const; |
|  | QRect SecCannonRect() const; |
|  | QRect shotRect() const; |
|  | QRect SecShotRect() const; |
|  |  |
|  | int currentAngle; |
|  | int currentForce; |
|  | int SecCurrentAngle; |
|  | int SecCurrentForce; |
|  |  |
|  | int timerCount; |
|  | int SecTimerCount; |
|  | QTimer \*autoShootTimer; |
|  | QTimer \*SecAutoShootTimer; |
|  | float shootAngle; |
|  | float shootForce; |
|  | float SecShootAngle; |
|  | float SecShootForce; |
|  |  |
|  | }; |
|  |  |
|  | #endif |

## ****lcdrange.h****

|  |
| --- |
| #ifndef LCDRANGE\_H |
|  | #define LCDRANGE\_H |
|  |  |
|  | #include <QWidget> |
|  |  |
|  | class QSlider; |
|  |  |
|  | class LCDRange : public QWidget |
|  | { |
|  | Q\_OBJECT |
|  |  |
|  | public: |
|  | LCDRange(QWidget \*parent = 0); |
|  |  |
|  | int value() const; |
|  |  |
|  | public slots: |
|  | void setValue(int value); |
|  | void setRange(int minValue, int maxValue); |
|  |  |
|  | signals: |
|  | void valueChanged(int newValue); |
|  | private: |
|  | QSlider \*slider; |
|  | }; |
|  |  |
|  | #endif |