Титульник

Оглавление

[1. ВВЕДЕНИЕ 5](#_Toc9377916)

[2. ПРЕДМЕТНАЯ ОБЛАСТЬ 5](#_Toc9377917)

[3. ТРЕБОВАНИЯ К ПРОГРАММЕ И ПО 6](#_Toc9377918)

[**3.1 Техническая часть 6**](#_Toc9377919)

[**3.2 Требования к интерфейсу программы 7**](#_Toc9377920)

[**3.3 Требования к составу и параметрам технических средств 7**](#_Toc9377921)

[**3.4 Требования к программным средствам, используемым программой 8**](#_Toc9377922)

[4.СДАЧА И ПРИЕМКА РАБОТЫ 8](#_Toc9377923)

[5.СТРУКТУРЫ ДАННЫХ И АЛГОРИТМЫ 8](#_Toc9377924)

[6.РЕАЛИЗАЦИЯ ПРОГРАММЫ 10](#_Toc9377925)

[7.UML-ДИАГРАММА 12](#_Toc9377926)

[8.ЗАКЛЮЧЕНИЕ 12](#_Toc9377927)

[9.СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ 13](#_Toc9377928)

[10. ПРИЛОЖЕНИЕ №1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ. 14](#_Toc9377929)

[1. ОБЩИЕ ПОЛОЖЕНИЯ 14](#_Toc9377930)

[1.1 Полное наименование системы и ее условное обозначение 14](#_Toc9377931)

[1.1.1 Полное наименование системы 14](#_Toc9377932)

[1.1.2 Краткое наименование системы 14](#_Toc9377933)

[1.2 Номер договора (контракта) 14](#_Toc9377934)

[1.3 Наименования организации-заказчика и организаций-участников работ 14](#_Toc9377935)

[1.3.1 Заказчик 14](#_Toc9377936)

[1.3.2 Исполнитель 14](#_Toc9377937)

[1.4 Перечень документов, на основании которых создается система 14](#_Toc9377938)

[1.5 Плановые сроки начала и окончания работы по созданию системы 14](#_Toc9377940)

[1.6 Источники и порядок финансирования работ 15](#_Toc9377941)

[1.7 Порядок оформления и предъявления заказчику результатов работ по созданию системы 15](#_Toc9377942)

[1.8 Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ 15](#_Toc9377943)

[1.9 Определения, обозначения и сокращения 15](#_Toc9377947)

[2 НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ 16](#_Toc9377948)

[2.1 Назначение системы 16](#_Toc9377949)

[2.2 Цели создания системы 16](#_Toc9377950)

[3 ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ 16](#_Toc9377956)

[4 ТРЕБОВАНИЯ К СИСТЕМЕ 16](#_Toc9377961)

[4.1 Требования к системе в целом 16](#_Toc9377962)

[4.1.1 Требования к структуре и функционированию системы 16](#_Toc9377963)

[4.1.1.1 Перечень подсистем, их назначение и основные характеристики 16](#_Toc9377964)

[4.1.1.2 Требования к способам и средствам связи для информационного обмена между компонентами системы 17](#_Toc9377969)

[4.1.2 Требования к численности и квалификации персонала системы 17](#_Toc9377970)

[4.1.3 Показатели назначения 17](#_Toc9377971)

[4.1.4 Требования к надежности 17](#_Toc9377972)

[4.1.5 Требования к безопасности 17](#_Toc9377973)

[4.1.6 Требования к эргономике и технической эстетике 18](#_Toc9377974)

[4.1.7 Требования к транспортабельности для подвижных АС 18](#_Toc9377975)

[4.1.8 Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы 18](#_Toc9377976)

[4.1.9 Требования к защите информации от несанкционированного доступа 18](#_Toc9377977)

[4.1.10 Требования по сохранности информации при авариях 18](#_Toc9377978)

[4.1.11 Требования к защите от влияния внешних воздействий 18](#_Toc9377979)

[4.1.12 Требования к патентной чистоте 19](#_Toc9377980)

[4.1.13 Требования по стандартизации и унификации 19](#_Toc9377981)

[4.1.14 Дополнительные требования 19](#_Toc9377982)

[4.2 Требования к видам обеспечения 19](#_Toc9377983)

[4.2.1 Требования к математическому обеспечению системы 19](#_Toc9377984)

[4.2.2 Требования информационному обеспечению системы 19](#_Toc9377985)

[4.2.3 Требования к лингвистическому обеспечению системы 19](#_Toc9377986)

[4.2.4 Требования к программному обеспечению системы 20](#_Toc9377987)

[4.2.5 Требования к техническому обеспечению 20](#_Toc9377988)

[4.2.6 Требования к организационному обеспечению 20](#_Toc9377989)

[4.2.7 Требования к методическому обеспечению 20](#_Toc9377990)

[5 СОСТАВ И СОДЕРЖАНИЕ РАБОТ ПО СОЗДАНИЮ (РАЗВИТИЮ) СИСТЕМЫ 20](#_Toc9377994)

[6 ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ СИСТЕМЫ 21](#_Toc9378002)

[6.1 Общие требования к приемке работ по стадиям 21](#_Toc9378003)

[7.ТРЕБОВАНИЯ К ДОКУМЕНТИРОВАНИЮ 22](#_Toc9378010)

[8.ИСТОЧНИКИ РАЗРАБОТКИ 22](#_Toc9378013)

[11. ПРИЛОЖЕНИЕ №2. ИСХОДНЫЙ КОД ПРОГРАММЫ. 22](#_Toc9378015)

# ВВЕДЕНИЕ

Язык C++ является одним из самых популярных языков программирования и имеет широкую область применения. Она включает 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. ТРЕБОВАНИЯ 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 начальной̆ 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 камней̆ 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. ОС Windows

4. Стандартный пакет С++;

## 3.4 Требования к программным средствам, используемым программой

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной ОС Windows 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.

# 5.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.

# 6.РЕАЛИЗАЦИЯ ПРОГРАММЫ

При запуске программы появится окно (рис. 1).

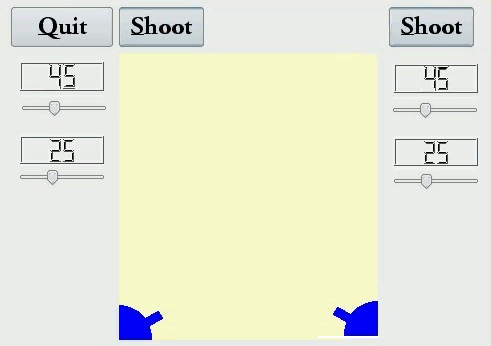


Рис.1 Начальный Экран

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

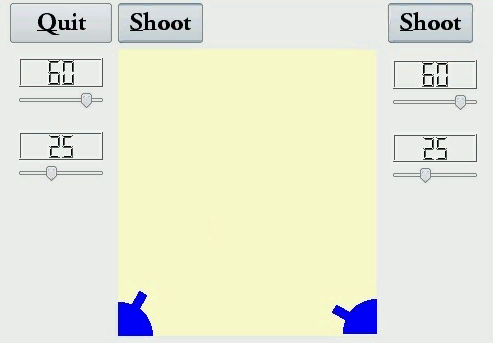


Рис.2 Изменение угла пушки

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

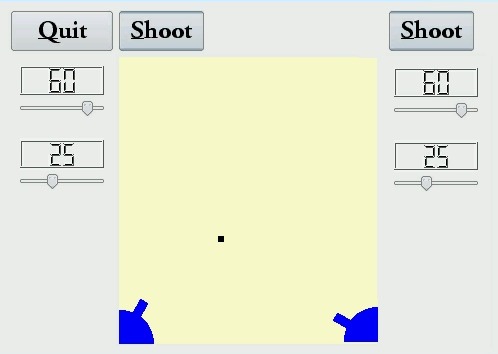
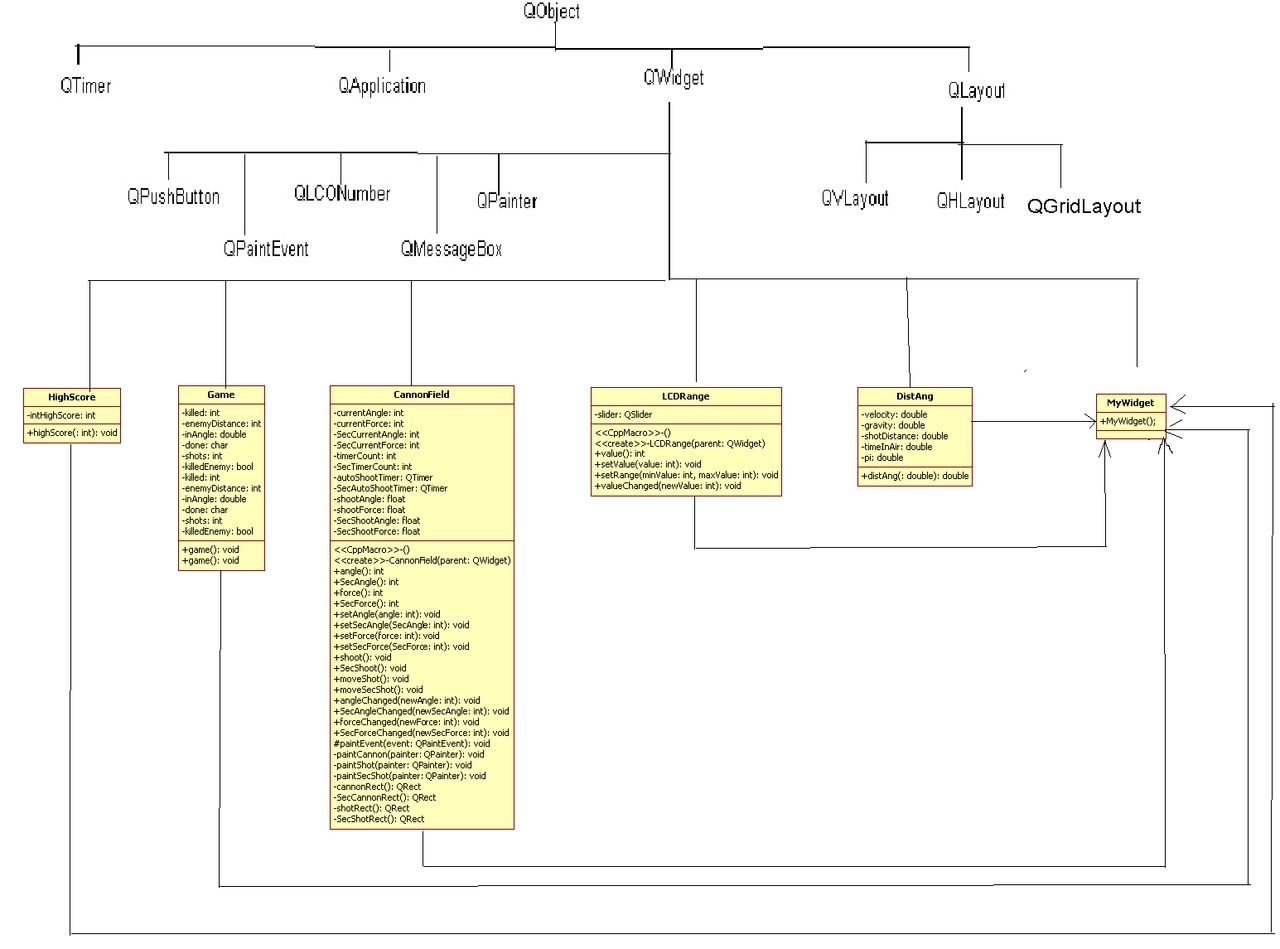


Рис. 3 Демонстрация стрельбы

Выход из игры реализуется с помощью кнопки «Выход».

# 7.UML-ДИАГРАММА



# 8.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 с мощной функциональностью, а использование графического интерфейса позволяет стороннему пользователю без труда использовать готовую систему.

# 9.СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

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

# 10. ПРИЛОЖЕНИЕ №1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ.

1. **ОБЩИЕ ПОЛОЖЕНИЯ** 
   1. **Полное наименование системы и ее условное обозначение**
      1. **Полное наименование системы**

Полное наименование: Простейшая Баллистическая игра

* + 1. **Краткое наименование системы**

Краткое наименование: Баллистическая игра

* 1. **Номер договора (контракта)**

Работа выполняется на основании Задания на выполнение курсовой работы.

* 1. **Наименования организации-заказчика и организаций-участников работ**
     1. **Заказчик**

Заказчик: РТУ МИРЭА

* + 1. **Исполнитель**

Исполнитель: студент группы ИНБО-04-18 Леоненков Александр Сергеевич

* 1. **Перечень документов, на основании которых создается система**

Документы, на основании которых создается система:

* Учебный план
  1. **Плановые сроки начала и окончания работы по созданию системы**

Плановый срок начала работы по созданию Баллистической игры – 26 февраля 2019 года.

Плановый срок окончания работы по созданию Баллистической игры – 20 мая 2019 года.

* 1. **Источники и порядок финансирования работ**

Разработка Системы финансируется РТУ МИРЭА, а, в частности, Федеральным бюджетом Российской Федерации.

* 1. **Порядок оформления и предъявления заказчику результатов работ по созданию системы**

Система передаётся в виде готового программного обеспечения на базе средств вычислительной техники Заказчика в сроки, установленные в п. 2.5 данного ТЗ. Приёмка осуществляется в составе Исполнителя и уполномоченных представителей Заказчика.

Порядок предъявления системы, её испытаний и окончательной приёмки определён в п.7 настоящего ТЗ. Совместно с предъявлением Системы Исполнителем производится сдача разработанного комплекта документации согласно п. 9 данного ТЗ.

* 1. **Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ**

При разработке Системы Исполнитель должен руководствоваться требованиями следующих документов:

* ГОСТ 34.601-90 Комплекс стандартов Автоматизированные системы. Стадии создания;
* Методические указания по выполнению курсовой работы для бакалавров.
* Федеральный закон об информации, информационных технологиях и о защите информации.
  1. **Определения, обозначения и сокращения**

Определения, обозначения и сокращения отсутствуют.

1. **НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ**
   1. **Назначение системы**

Система предназначена для моделирования броска камня, траектории его полета и следит за тем, попал ли в цель камень.

* 1. **Цели создания системы**

Цели создания Системы:

* Демонстрация программы, которая может построить и смоделировать бросок, не пренебрегая законам физики;
* сдача курсовой работы;
* приобретение опыта разработки приложений средней сложности;
* изучение объектно-ориентированного программирования;
* приобретение навыков написания технической документации.

1. **ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ**

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

1. **ТРЕБОВАНИЯ К СИСТЕМЕ**
   1. **Требования к системе в целом**
      1. **Требования к структуре и функционированию системы**
         1. **Перечень подсистем, их назначение и основные характеристики**

Разрабатываемая Баллистическая игра должна состоять из нескольких классов, а именно:

* CannonField
* LCDRange
* MyWidget
* DistAng
  + - 1. **Требования к способам и средствам связи для информационного обмена между компонентами системы**

Связь между компонентами Баллистической игры обеспечивается функциями, определёнными в компонентах Системы.

* + 1. **Требования к численности и квалификации персонала системы**

Для работы разрабатываемой Баллистической игры необходимо и достаточно одного человека. Данная Система не требует какой-либо квалификации или подготовки пользователей для работы с ней.

* + 1. **Показатели назначения**

Показателями назначения Баллистической игры являются результаты, совпадающие с ожидаемыми во время тестирования.

* + 1. **Требования к надежности**

Данная Баллистическая игра должна хранить все данные в динамической памяти Исключения составляют элементы графического интерфейса, которые должны храниться в отдельном каталоге, находящемся в том же каталоге, что и Баллистическая игра

* + 1. **Требования к безопасности**

Работу данной Баллистической игры следует завершать только по нажатии специальной кнопки «Quit» в левом верхнем углу графического интерфейса.

* + 1. **Требования к эргономике и технической эстетике**

Графический интерфейс Баллистической игры должен разрабатываться на основе офтальмологических исследований, с использованием сочетаний цветов, комфортных для глаз человека.

* + 1. **Требования к транспортабельности для подвижных АС**

Разрабатываемая Баллистическая игра должна запускаться на любых устройствах под управлением операционной системы не старше чем Windows XP без процедуры установки или повторной компиляции данной Баллистической игры.

* + 1. **Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы**

Для корректной работы разрабатываемая Баллистическая игра должна храниться в отдельном каталоге со всеми своими компонентами. Особых требований по обслуживанию данная Баллистическая игра не имеет.

* + 1. **Требования к защите информации от несанкционированного доступа**

Данная Баллистическая игра должна обеспечивать только динамическое хранение информации.

* + 1. **Требования по сохранности информации при авариях**

В случае аварийной ситуации текущее состояние Баллистической игры можно получить, введя те же самые входные данные.

* + 1. **Требования к защите от влияния внешних воздействий**

Разрабатываемая Баллистическая игра должна обеспечивать безопасную работу без случайных или умышленных изменений, внесённых при переключении между состояниями. Для того, чтобы изменить состояние Баллистической игры вручную, требуется предварительно переключить её в режим ожидания ввода информации и лишь за тем дополнять или изменять состояние поля.

* + 1. **Требования к патентной чистоте**

Данная Баллистическая игра должна являться интеллектуальной собственностью Исполнителя и быть патентно чистой по отношению ко всем странам мира.

* + 1. **Требования по стандартизации и унификации**

Данная Баллистическая игра не должна использовать простейший способ реализации с использованием матрицы, хранящей состояние всех клеток поля. Баллистическая игра также должна быть реализована методами объектно-ориентированного программирования.

* + 1. **Дополнительные требования**

Разрабатываемая Баллистическая игра дополнительных требований не имеет.

* 1. **Требования к видам обеспечения**
     1. **Требования к математическому обеспечению системы**

Разрабатываемая Баллистическая игра должна уметь высчитать размеры окна, исходя из разрешения экрана, отмасштабировать элементы графического интерфейса под размеры окна, построить сетку, изображающую поле, а также рассчитать и смоделировать траекторию полета.

* + 1. **Требования информационному обеспечению системы**

Баллистическая игра должна получать на вход размеры поля в количестве клеток и начальное состояние системы. Также Баллистическая игра может получать сигналы, контролирующие её работу.

* + 1. **Требования к лингвистическому обеспечению системы**

Для создания Баллистической игры должен быть использован низкоуровневый язык программирования C++, а также фреймворк QT. Для взаимодействия Баллистической игры с пользователем, руководства по использованию Баллистической игры и документации Баллистической игры должен использоваться русский язык.

* + 1. **Требования к программному обеспечению системы**

Данная Баллистическая игра должна являться кроссплатформенным приложением и работать на всех операционных системах с поддержкой графического интерфейса. Для корректной работы Системы не требуется загрузки сторонних приложений или библиотек.

* + 1. **Требования к техническому обеспечению**

Для корректной работы Баллистической игры, она должна запускаться на устройствах с подключённым экраном, мышью и клавиатурой. Также ширина экрана должна быть строго больше чем его высота.

* + 1. **Требования к организационному обеспечению**

Разрабатываемая Баллистическая игра не должна начинать работу при неверно введённых входных данных. В этом случае она должна либо проигнорировать ввод запрещённых символов, либо выводить информационное окно с пояснением ошибки ввода.

* + 1. **Требования к методическому обеспечению**

Данная система должна поставляться с определённым пакетом документации, состоящем из:

* технического задания (ГОСТ 34.602-89);
* отчёта по выполнению курсовой работы;
* задания на выполнение курсовой работы.

1. **СОСТАВ И СОДЕРЖАНИЕ РАБОТ ПО СОЗДАНИЮ (РАЗВИТИЮ) СИСТЕМЫ**

* Сбор необходимой информации

Результат: определение целей, задач проектируемой системы.

* Анализ предметной области

Результат: введение базовых требований к решению задач и целей.

* Разработка ТЗ

Результат: готовое техническое задание с определёнными сроками выполнения курсовой работы.

* Разработка модели программы

Результат: полностью спроектированные классы, готовая UML диаграмма.

* Разработка готового проекта

Результат: готовая рабочая программа.

* Тестирование программного продукта

Результат: устранение ошибок и недочётов в работе программного продукта.

* Сдача системы в эксплуатацию с описанием алгоритмов и готовой технической документацией

1. **ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ СИСТЕМЫ**
   1. **Общие требования к приемке работ по стадиям**

Разработка данной Баллистической игры делится на шесть стадий:

* получение задания на выполнение курсовой работы;
* составление и согласование технического задания;
* создание и тестирование Баллистической игры Исполнителем;
* написание технической документации для Баллистической игры;
* демонстрация Баллистической игры Заказчику;
* защита курсовой работы.

**6.ТРЕБОВАНИЯ К СОСТАВУ И СОДЕРЖАНИЮ РАБОТ ПО ПОДГОТОВКЕ ОБЪЕКТА АВТОМАТИЗАЦИИ К ВВОДУ СИСТЕМЫ В ДЕЙСТВИЕ**

Для корректной работы разрабатываемой Баллистической игры необходимо использовать экраны, ширина которых строго больше их высоты.

**7.ТРЕБОВАНИЯ К ДОКУМЕНТИРОВАНИЮ**

Кроме создания работоспособной Баллистической игры Исполнитель должен составить пакет документации, состоящий из:

* технического задания;
* пояснительной записки.

**8.ИСТОЧНИКИ РАЗРАБОТКИ**

В качестве источников разработки использовались данные ресурсы:

* Зорина, Н.В. Методические указания по выполнению курсовой работы для бакалавров, обучающихся по направлениям 09.03.02 «Информационные системы и технологии», 09.03.04 «Программная инженерия» / Н.В. Зорина, Л.Б. Зорин, О.В. Соболев,- Москва, 2017 - 41 с.

# 11. ПРИЛОЖЕНИЕ №2. 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 |