ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7  
по дисциплине «Человеко-машинное взаимодействие»

Выполнил: студент гр. ИП-813  
Бурдуковский И.А.

Проверил:   
Мерзлякова Е.Ю

Новосибирск 2020

**Цель работы:**

Научиться работать с графикой в Qt.

**Задание:**

1. Создать графическую сцену.
2. Поместить на сцену различные элементы для составления картинки по варианту. Обязательно использовать и геометрические фигуры, и картинки. Они должны перемещаться с помощью мыши.
3. Ограничить края сцены «стенами» в виде каких-либо элементов.
4. Поместить на сцену движущийся элемент по заданию. Он должен перемещаться с заданной скоростью, сталкиваться со «стенами» и фигурами на сцене. Используйте таймер и функцию обнаружения столкновений.

По варианту необходимо:

Машина и движущийся мяч

**Листинг программы:**

main.cpp:

#include "widget\_burdukovskiy.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

widget\_burdukovskiy w;

w.show();

return a.exec();

}

widget\_burdukovskiy.cpp:

#include "widget\_burdukovskiy.h"

#include "ui\_widget\_burdukovskiy.h"

widget\_burdukovskiy::**widget\_burdukovskiy**(QWidget \*parent) : QWidget(parent), ui(new Ui::widget\_burdukovskiy)

{

ui->setupUi(this);

this->resize(530,530); /// Задаем размеры виджета, то есть окна

this->setFixedSize(530,530); /// Фиксируем размеры виджета

scene = new QGraphicsScene(); /// Инициализируем графическую сцену

Ball = new ball(); /// Инициализируем мяч

Car = new car(); ///Инициализируем машину

ui->graphicsView->setScene(scene); /// Устанавливаем графическую сцену в graphicsView

ui->graphicsView->setRenderHint(QPainter::Antialiasing); /// Устанавливаем сглаживание

ui->graphicsView->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff); /// Отключаем скроллбар по вертикали

ui->graphicsView->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff); /// Отключаем скроллбар по горизонтали

scene->setSceneRect(-250,-250,500,500); /// Устанавливаем область графической сцены

/\* Дополнительно нарисуем органичение территории в графической сцене \*/

scene->addLine(-250,-250, 250,-250, QPen(Qt::black));

scene->addLine(-250, 250, 250, 250, QPen(Qt::black));

scene->addLine(-250,-250,-250, 250, QPen(Qt::black));

scene->addLine( 250,-250, 250, 250, QPen(Qt::black));

scene->addItem(Ball); /// Добавляем на сцену мяч

scene->addItem(Car); ///Добавляем на сцену машину

Ball->setPos(200,0); /// Устанавливаем мяч в центр сцены

Car->setPos(0, 0); /// Устанавливаем машину в центр сцены

/\* Инициализируем таймер и вызываем слот обработки сигнала таймера

\* у мяча 20 раз в секунду.

\* Управляя скоростью отсчётов, соответственно управляем скоростью

\* изменения состояния графической сцены. Все это нужно для автоматического движения.

\* \*/

timer = new QTimer();

connect(timer, &QTimer::timeout, Ball, &ball::slotGameTimer);

timer->start(1000 / 50);

}

widget\_burdukovskiy::~***widget\_burdukovskiy***()

{

delete ui;

}

car.cpp:

#include "car.h"

car::**car**(QObject \*parent) :

QObject(parent), QGraphicsItem()

{

}

car::~***car***()

{

}

QRectF car::***boundingRect***() const

{

QImage source;

source.load("D:\\labs\\qt\\7\\lub7\\car.png");

QRectF tmp(source.rect());

return tmp; /// Ограничиваем область, в которой лежит наша машинка

}

void car::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

/\*отрисовка машины\*/

QImage source;

source.load("D:\\labs\\qt\\7\\lub7\\car.png");

QRect rect(source.rect());

painter->drawImage(rect, source);

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

void car::***mouseMoveEvent***(QGraphicsSceneMouseEvent \*event)

{

/\* Устанавливаем позицию графического элемента

\* в графической сцене, транслировав координаты

\* курсора внутри графического элемента

\* в координатную систему графической сцены

\* \*/

this->setPos(mapToScene(event->pos()));

}

void car::***mousePressEvent***(QGraphicsSceneMouseEvent \*event)

{

/\* При нажатии мышью на графический элемент

\* заменяем курсор на руку, которая держит этот элемента

\* \*/

this->setCursor(QCursor(Qt::ClosedHandCursor));

Q\_UNUSED(event);

}

void car::***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event)

{

/\* При отпускании мышью элемента

\* заменяем на обычный курсор стрелку

\* \*/

this->setCursor(QCursor(Qt::ArrowCursor));

Q\_UNUSED(event);

}

ball.cpp:

#include "ball.h"

ball::**ball**(QObject \*parent) :

QObject(parent), QGraphicsItem()

{

UP = true;

DOWN = false;

LEFT = true;

RIGHT = false;

}

ball::~***ball***()

{

}

QRectF ball::***boundingRect***() const

{

return QRectF(-30,-30,30,30); /// Ограничиваем область, в которой лежит мячик

}

void ball::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget)

{

painter->setBrush(Qt::red); /// Устанавливаем кисть, которой будем отрисовывать объект

painter->drawEllipse(-30, -30, 30, 30); /// Рисуем мячик

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

void ball::***mouseMoveEvent***(QGraphicsSceneMouseEvent \*event)

{

/\* Устанавливаем позицию графического элемента

\* в графической сцене, транслировав координаты

\* курсора внутри графического элемента

\* в координатную систему графической сцены

\* \*/

this->setPos(mapToScene(event->pos()));

}

void ball::***mousePressEvent***(QGraphicsSceneMouseEvent \*event)

{

/\* При нажатии мышью на графический элемент

\* заменяем курсор на руку, которая держит этот элемента

\* \*/

this->setCursor(QCursor(Qt::ClosedHandCursor));

Q\_UNUSED(event);

}

void ball::***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event)

{

/\* При отпускании мышью элемента

\* заменяем на обычный курсор стрелку

\* \*/

this->setCursor(QCursor(Qt::ArrowCursor));

Q\_UNUSED(event);

}

void ball::**slotGameTimer**()

{

/\* Поочерёдно выполняем проверку на нажатие клавиш

\* с помощью функции асинхронного получения состояния клавиш,

\* которая предоставляется WinAPI

\* \*/

if(LEFT == true){

setPos(mapToParent(-5, 0));

}

if(RIGHT == true){

setPos(mapToParent(5, 0));

}

if(UP == true){

setPos(mapToParent(0, -5)); /\* Продвигаем объект на 5 пискселей вперёд

\* перетранслировав их в координатную систему

\* графической сцены

\* \*/

}

if(DOWN == true){

setPos(mapToParent(0, 5)); /\* Продвигаем объект на 5 пискселей назад

\* перетранслировав их в координатную систему

\* графической сцены

\* \*/

}

/\* Проверка выхода за границы поля

\* Если объект выходит за заданные границы, то возвращаем его назад, сменяя направление полёта через булевские переменные

\* \*/

if(this->x() - 10 < -230){

this->setX(-220);// слева

LEFT = false;

RIGHT = true;

}

if(this->x() + 10 > 255){

this->setX(245); // справа

RIGHT = false;

LEFT = true;

}

if(this->y() - 10 < -230){

this->setY(-220); // сверху

UP = false;

DOWN = true;

}

if(this->y() + 10 > 255){

this->setY(245); // снизу

UP = true;

DOWN = false;

}

}

widget\_burdukovskiy.h:

#ifndef WIDGET\_BURDUKOVSKIY\_H

#define WIDGET\_BURDUKOVSKIY\_H

#include <QWidget>

#include <QGraphicsScene>

#include <QShortcut>

#include <QTimer>

#include "ball.h"

#include "car.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class widget\_burdukovskiy; }

QT\_END\_NAMESPACE

class widget\_burdukovskiy : public QWidget

{

Q\_OBJECT

public:

explicit widget\_burdukovskiy(QWidget \*parent = 0);

~***widget\_burdukovskiy***();

private:

Ui::widget\_burdukovskiy \*ui;

QGraphicsScene \*scene;

ball \*Ball;

car \*Car;

QTimer \*timer;

};

#endif // WIDGET\_BURDUKOVSKIY\_H

car.h:

#ifndef CAR\_H

#define CAR\_H

#include <QObject>

#include <QGraphicsItem>

#include <QPainter>

#include <QGraphicsSceneMouseEvent>

#include <QDebug>

#include <QCursor>

#include <QPixmap>

/\* Подключаем библиотеку, отвечающую за использование WinAPI

\* Данная библиотека необходима для асинхронной проверки состояния клавиш

\* \*/

#include <windows.h>

class car : public QObject, public QGraphicsItem

{

Q\_OBJECT

public:

explicit car(QObject \*parent = 0);

~***car***();

protected:

QRectF ***boundingRect***() const;

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget);

private:

void ***mouseMoveEvent***(QGraphicsSceneMouseEvent \*event);

void ***mousePressEvent***(QGraphicsSceneMouseEvent \*event);

void ***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event);

};

#endif // CAR\_H

ball.h:

#ifndef BALL\_H

#define BALL\_H

#include <QObject>

#include <QGraphicsItem>

#include <QPainter>

#include <QGraphicsSceneMouseEvent>

#include <QDebug>

#include <QCursor>

class ball: public QObject, public QGraphicsItem

{

Q\_OBJECT

public:

explicit ball(QObject \*parent = 0);

~***ball***();

signals:

public slots:

void **slotGameTimer**(); // Слот, который отвечает за обработку перемещения мячика

protected:

QRectF ***boundingRect***() const; // получение квадрата мячика

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget);//отрисовка мячика

private:

bool LEFT; //состояния полёта

bool RIGHT;

bool UP;

bool DOWN;

QRect rect;

QTimer \*timer;

void ***mouseMoveEvent***(QGraphicsSceneMouseEvent \*event);//методы для пермещения мячика мышкой

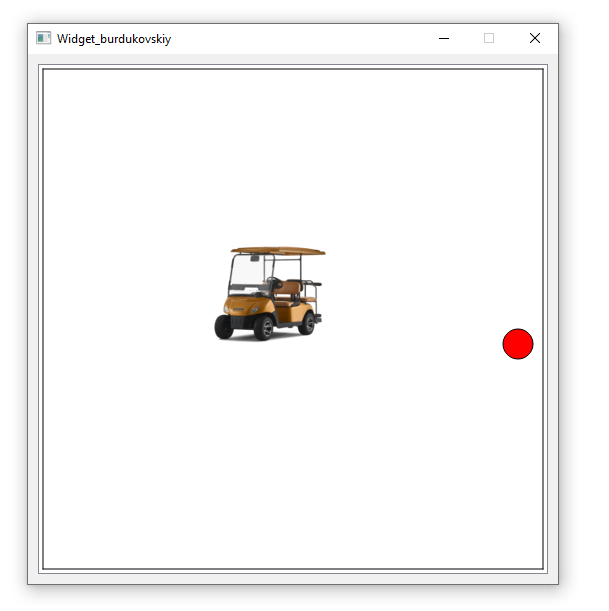
void ***mousePressEvent***(QGraphicsSceneMouseEvent \*event);

void ***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event);

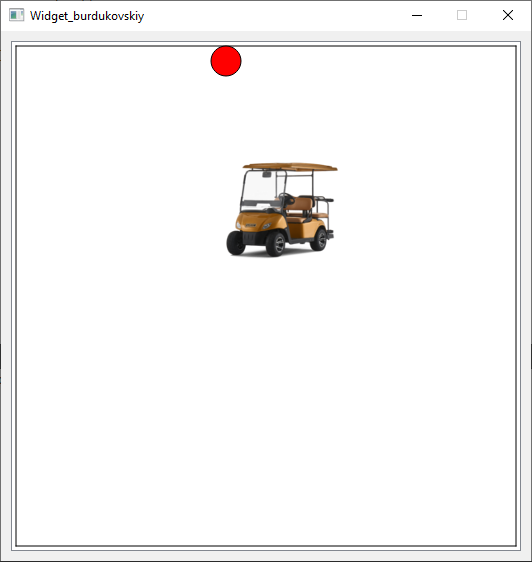
};

#endif // BALL\_H

**Результат работы программы:**



На скриншоте виден мяч, картинка машины и стены, от которых отталкивается мяч



На скриншоте видно передвижение мяча и машины