**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИИ**  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**  
**(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №3

Компьютерная графика

**Аффинные преобразования в плоскости**

Выполнил: ст. группы ВТ-22  
Ковалёв И. Д.

Проверил: Осипов О. В.

**Белгород 2019**

**Цель работы:**

Получение навыков выполнения преобразований на плоскости и создание графического приложения с использованием GDI в среде Qt Creator.

**Порядок выполнения работы**

1. Разработать алгоритм и составить программу для построения на экране изображения в соответствии с номером варианта. В качестве исходных данных взять указанные в таблице №1.

**Требования к программе**

1. Разработать модуль для выполнения аффинных преобразований на плоскости с помощью матриц. В модуле должны быть реализованы перегруженные операции действия с матрицами (умножение), с векторами и матрицами (умножение вектора-строки на матрицу),конструкторы различных матриц (переноса, масштабирования, переноса, отражения).
2. В программе должна быть предусмотрена возможность ввода пользователем исходных данных (из правой колонки таблицы №1).
3. Разбить окно на 2 равные части. В левой части должна выводиться основная анимация, в правой части её отражение относительно вертикальной линии, проходящей через центр окна.
4. Изображение должно масштабироваться по центру левой и правой части окна с отступом 10 пикселей от границ и вертикальной линии и реагировать на изменение размера окна (см. пример проекта lab\_1\_CSharp).
5. Раскрасить (залить) примитивы (круги, многоугольники и др.) по собственному усмотрению.

Содержимое файла mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QMainWindow>

#include <QPainter>

#include <QTimer>

#include "matrix.h"

#include "vector.h"

#include "circle.h"

#include "stamp.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

void paintEvent(QPaintEvent \*e);

void drawStamp(QPainter \* painter);

void drawConv(QPainter \* painter);

void drawLowerCircles(QPainter \* painter);

void drawMainCircles(QPainter \* painter);

private:

Ui::MainWindow \*ui;

QTimer \*timer;

int tickSkip;

int radius;

int angle;

int scaledCircleWidth;

Vector center;

QPolygon conv, ltr, trt;

Stamp stamp;

QVector <Circle \*> circles;

QRect lowerCircles[3], circleStartPos, circleReadyPos;

private slots:

void animate();

};

#endif // MAINWINDOW\_H

Содержимое файла mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QtDebug>

MainWindow::~***MainWindow***(){

delete ui;

}

void MainWindow::**animate**(){

update();

}

MainWindow::**MainWindow**(QWidget \*parent):QMainWindow(parent), ui(new Ui::MainWindow){

ui->setupUi(this);

this->setMinimumSize(1200, 600);

this->setMaximumSize(1200, 600);

angle = 0;

timer = new QTimer(this);

timer->setInterval(1000/360);

//timer->setInterval(1000/4);

connect(timer, SIGNAL(timeout()), this, SLOT(animate()));

timer->start();

radius = this->height() / 20;

scaledCircleWidth = radius;

center = Vector(this->width() / 4, this->height() / 2);

tickSkip = 3;

conv.push\_back(QPoint(0, center.y() ));

conv.push\_back(QPoint(0, center.y() - (radius \* 2) ));

conv.push\_back(QPoint(radius \* 2, center.y() - (radius \* 2) ));

conv.push\_back(QPoint(radius \* 2, center.y() ));

conv.push\_back(QPoint(this->width() / 3, center.y() ));

conv.push\_back(QPoint(this->width() / 3, center.y() + radius ));

conv.push\_back(QPoint(0, center.y() + radius ));

ltr.push\_back(QPoint(-50, -50));

ltr.push\_back(QPoint(-50, 50));

ltr.push\_back(QPoint(50, 50));

int tmpx = (ltr[0].x() + ltr[1].x() + ltr[2].x() )/ 3;

int tmpy = (ltr[0].y() + ltr[1].y() + ltr[2].y() )/ 3;

QPolygon newltr;

Vector tmp;

for (int i = 0; i < 3; i++){

tmp.setPoint(ltr[i]);

tmp = Matrix::translationMatrix(-tmpx, -tmpy) \* tmp;

newltr.push\_back(tmp.getPoint());

}

ltr = newltr;

stamp = Stamp(QPoint(this->width() / 3, center.y() + radius), QPoint(this->width() / 3 + radius \* 2, 0));

lowerCircles[0] = QRect(this->width() / 4 - radius, center.y() + radius, radius, radius);

QPoint tr = QPoint(lowerCircles[0].center().x() - radius / 8, lowerCircles[0].center().y() + radius / 8);

QPoint bl = QPoint(tr.x() - 2 \* radius, tr.y() + 2 \* radius);

QPoint tl = QPoint(bl.x(), tr.y());

QPoint br = QPoint(tr.x(), bl.y());

lowerCircles[1] = QRect(tl, br);

tl = QPoint(tl.x() - 1.5 \* radius, tl.y() + 1.5 \*radius);

br = QPoint(br.x() - 1.5 \*radius, br.y() + 1.5 \*radius);

lowerCircles[2] = QRect(tl, br);

tl = QPoint(0, center.y());

br = QPoint(radius \* 2, center.y() - radius \* 2);

circleStartPos = QRect(tl, br);

tl = QPoint(this->width() / 3 , center.y() -2 \* radius);

br = QPoint(tl.x() + 2 \* radius , tl.y() + 2 \* radius);

circleReadyPos = QRect(tl, br);

Circle \* firstCircle = new Circle(circleStartPos);

circles.push\_back(firstCircle);

}

void MainWindow::***paintEvent***(QPaintEvent \*e){

QPainter painter(this);

painter.setBrush(Qt::black);

painter.drawLine(QPointF(this->width() / 2.0, 0), QPointF(this->width() / 2.0, this->height()));

drawStamp(&painter);

drawLowerCircles(&painter);

drawMainCircles(&painter);

drawConv(&painter);

QPolygon newltr;

Vector tmp;

//Вращение треугольника

/\*for (int i = 0; i < 3; i++){

tmp.setPoint(ltr[i]);

tmp = Matrix::rotationMatrix(angle) \* tmp;

//qDebug() << "angle : " << angle;

tmp = Matrix::translationMatrix(100, 100) \* tmp;

newltr.push\_back(tmp.getPoint());

}

//qDebug() << newltr;

painter.setBrush(QColor(3, 219, 252));

painter.drawPolygon(newltr);

\*/

painter.setBrush(Qt::black);

painter.end();

}

void MainWindow::**drawConv**(QPainter \* painter){

painter->setPen(Qt::black);

painter->drawPolygon(conv);

QPolygon convReflection;

Vector tmp;

for (int i = 0; i < conv.size(); i++){

tmp = Matrix::reflectionMatrix(0) \* Vector(conv[i].x(), conv[i].y());

tmp = Vector(tmp.x() + this->width(), tmp.y());

convReflection.push\_back(QPoint(tmp.x(), tmp.y()));

}

painter->drawPolygon(convReflection);

}

void MainWindow::**drawLowerCircles**(QPainter \* painter){

QBrush oldBrush = painter->brush();

if (angle < 360) angle++; else angle = 0;

int f = 1;

for (int i = 0; i < 3; i++){

f \*= -1;

painter->setBrush(Qt::red);

painter->drawPie(lowerCircles[i], (0 + angle) \* 16 \* f, 90 \* 16);

painter->setBrush(Qt::blue);

painter->drawPie(lowerCircles[i], (90 + angle) \* 16 \* f, 90 \* 16);

painter->setBrush(Qt::red);

painter->drawPie(lowerCircles[i], (180 + angle) \* 16 \* f, 90 \* 16);

painter->setBrush(Qt::blue);

painter->drawPie(lowerCircles[i], (270 + angle) \* 16 \* f, 90 \* 16);

}

Vector tl, br;

f \*= -1;

for (int i = 0; i < 3; i++){

f \*= -1;

tl = Matrix::reflectionMatrix(0) \* Vector(lowerCircles[i].topLeft().x(), lowerCircles[i].topLeft().y());

br = Matrix::reflectionMatrix(0) \* Vector(lowerCircles[i].bottomRight().x(), lowerCircles[i].bottomRight().y());

QRect reflectedRect = QRect(QPoint(tl.x() + this->width(), tl.y()), QPoint(br.x() + this->width(), br.y()));

painter->setBrush(Qt::red);

painter->drawPie(reflectedRect, (0 - angle) \* 16 \* f, 90 \* 16);

painter->setBrush(Qt::blue);

painter->drawPie(reflectedRect, (90 - angle) \* 16 \* f, 90 \* 16);

painter->setBrush(Qt::red);

painter->drawPie(reflectedRect, (180 - angle) \* 16 \* f, 90 \* 16);

painter->setBrush(Qt::blue);

painter->drawPie(reflectedRect, (270 - angle) \* 16 \* f, 90 \* 16);

}

painter->setBrush(oldBrush);

}

void MainWindow::**drawMainCircles**(QPainter \*painter){

QBrush oldBrush = painter->brush();

bool newCircleNeeded = false;

for (int i = 0; i < circles.size(); i++){

if ((circles[i]->getRect().bottomLeft().x() == circleReadyPos.bottomLeft().x()) && !circles[i]->isReady()){

circles[i]->setDirection(0);

circles[i]->setReadyPos(1);

}

circles[i]->move();

//Определяем, нужно ли уменьшать шар

if (circles[i]->isReady() && circles[i]->getWidth() != scaledCircleWidth){

if (!tickSkip){

circles[i]->scale();

tickSkip = 10;

}

tickSkip--;

}

painter->setBrush(Qt::red);

painter->drawEllipse(circles[i]->getRect());

Vector tl ,br;

tl.setPoint(circles[i]->getRect().topLeft());

br.setPoint(circles[i]->getRect().bottomRight());

tl = Matrix::reflectionMatrix(0) \* tl;

br = Matrix::reflectionMatrix(0) \* br;

//QPoint tmp1(this->width() + tl.x(), tl.y()), tmp2(this->width() + br.x(), br.y());

QPoint tmp1(this->width() + tl.x(), tl.y()), tmp2(this->width() + br.x(), br.y());

//QRect circleReflection = QRect(tl.getPoint(), br.getPoint());

QRect circleReflection = QRect(tmp1, tmp2);

painter->drawEllipse(circleReflection);

//qDebug() << circleReflection;

}

if (circles[circles.size() - 1]->getRect().bottomLeft().y() == 335){

Circle \* tmpCircle = new Circle(circleStartPos);

circles.push\_back(tmpCircle);

newCircleNeeded = false;

}

//Если очередной шар достиг нижней границы экрана - удалить

if (circles[circles.size() - 1]->getRect().bottomRight().y() == 600) {

delete circles[circles.size() - 1];

circles.pop\_front();

circles.resize(circles.size() - 2);

}

//qDebug() << circles.size() << " - количество шаров";

painter->setBrush(oldBrush);

}

void MainWindow::**drawStamp**(QPainter \* painter){

QBrush oldBrush = painter->brush();

painter->setBrush(Qt::black);

//Сдивигаем штамп

stamp.move();

if (stamp.getLowerPos() == -240 || stamp.getLowerPos() == 0) stamp.changeDirection();

//Получаем отражение штампа

QRect stampReflection;

Vector tmp1, tmp2;

tmp1 = Matrix::reflectionMatrix(0) \* Vector(stamp.getRect().topLeft().x() - this->width(), stamp.getRect().topLeft().y());

tmp2 = Matrix::reflectionMatrix(0) \* Vector(stamp.getRect().bottomRight().x() - this->width(), stamp.getRect().bottomRight().y());

stampReflection = QRect(tmp1.getPoint(), tmp2.getPoint());

//Выводим красоту на экран

painter->drawRect(stamp.getRect());

painter->drawRect(stampReflection);

painter->setBrush(oldBrush);

}

Содержимое файла circle.h:

#ifndef CIRCLE\_H

#define CIRCLE\_H

#include <QRect>

#include <QPainter>

#include <vector.h>

#include <matrix.h>

class **Circle**{

QRect rect;

int direction;

int inc;

bool reachedReadyPos;

bool scalingNeeded;

public:

**Circle**();

**Circle**(QRect r);

~**Circle**(){};

QRect **getRect**();

//0 - вдоль оси х, 1 - вдоль оси у; inc - знак приращения, для отражения -1:

void **move**();

void **draw**(QPainter \* painter);

void **scale**();

int **getDirection**();

int **getInc**();

int **getWidth**();

void **setReadyPos**(bool);

void **setDirection**(int newDirection);

void **setInc**(int newInc);

bool **isReady**();

bool **isScalingNeeded**();

};

#endif // CIRCLE\_H

Содержимое файла circle.cpp:

#include "circle.h"

#include "QDebug"

Circle::Circle(){

rect = QRect();

direction = 1;

inc = 1;

reachedReadyPos = 0;

}

Circle::Circle(QRect r){

rect = r;

direction = 1;

inc = 1;

reachedReadyPos = 0;

}

QRect Circle::getRect(){

return rect;

}

void Circle::move(){

QPoint tl, br;

if (direction){

tl = QPoint(rect.topLeft().x() + 1 \* inc, rect.topLeft().y());

br = QPoint(rect.bottomRight().x() + 1 \* inc, rect.bottomRight().y());

rect = QRect(tl, br);

} else {

tl = QPoint(rect.topLeft().x(), rect.topLeft().y() + 1 \* inc);

br = QPoint(rect.bottomRight().x(), rect.bottomRight().y() + 1 \* inc);

rect = QRect(tl, br);

}

}

void Circle::draw(QPainter \*painter){

QBrush prevBrush = painter->brush();

painter->setBrush(Qt::red);

painter->drawEllipse(rect);

painter->setBrush(prevBrush);

}

void Circle::scale(){

Vector tl, br;

QPoint center = rect.center();

//qDebug() << rect;

tl = Matrix::scalingMatrix(0.9, 0.9) \* Vector(rect.topLeft().x(), rect.topLeft().y());

br = Matrix::scalingMatrix(0.9, 0.9) \* Vector(rect.bottomRight().x(), rect.bottomRight().y());

//qDebug() << tl.x() << tl.y();

//qDebug() << br.x() << br.y();

rect = QRect(tl.getPoint(), br.getPoint());

int rx, ry;

rx = center.x() - rect.center().x();

ry = center.y() - rect.center().y();

tl = Matrix::translationMatrix(rx, ry) \* tl;

br = Matrix::translationMatrix(rx, ry) \* br;

rect = QRect(tl.getPoint(), br.getPoint());

//qDebug() << rect;

}

int Circle::getDirection(){

return direction;

}

void Circle::setDirection(int newDirection){

direction = newDirection;

}

int Circle::getInc(){

return inc;

}

int Circle::getWidth(){

return rect.bottomRight().x() - rect.bottomLeft().x();

}

void Circle::setInc(int newInc){

inc = newInc;

}

void Circle::setReadyPos(bool f){

reachedReadyPos = f;

}

bool Circle::isReady(){

return reachedReadyPos;

}

bool Circle::isScalingNeeded(){

return scalingNeeded;

}

Содержимое файла stamp.h:

#ifndef STAMP\_H

#define STAMP\_H

#include <QPolygon>

#include <matrix.h>

class Stamp{

QRect s;

int direction;

int translation;

public:

Stamp();

Stamp(QRect newStamp):s(newStamp), direction(1), translation(0){}

Stamp(QPoint tl, QPoint br);

void move();

void changeDirection();

void setDirection(int newDirection);

int getLowerPos();

int getTranslation();

QRect getRect();

};

#endif // STAMP\_H

Содержимое файла stamp.cpp:

#include "stamp.h"

#include <QDebug>

Stamp::Stamp(){

direction = -1;

translation = -1;

}

Stamp::Stamp(QPoint tl, QPoint br){

s = QRect(tl, br);

direction = -1;

translation = -1;

}

void Stamp::move(){

Vector tmp, tmp1;

tmp = Matrix::translationMatrix(0, translation) \* Vector(s.topLeft().x(), s.topLeft().y());

tmp1= Matrix::translationMatrix(0, translation) \* Vector(s.bottomRight().x(), s.bottomRight().y());

s = QRect(tmp.getPoint(), tmp1.getPoint());

}

int Stamp::getLowerPos(){

return s.bottomLeft().y();

}

QRect Stamp::getRect(){

return s;

}

void Stamp::changeDirection(){

direction \*= -1;

translation \*= -1;

}

void Stamp::setDirection(int newDirection){

direction = newDirection;

}

int Stamp::getTranslation(){

return translation;

}

Содержимое файла matrix.h:

#ifndef MATRIX\_H

#define MATRIX\_H

#include "vector.h"

#include <cmath>

#include <QTransform>

class Matrix{

public:

double m[3][3];

Matrix();

Matrix(const Matrix &newMatrix);

Matrix& operator = (const Matrix &newMatrix);

static Matrix rotationMatrix(double angle);

static Matrix scalingMatrix(double kx, double ky);

//0 - ось Х, 1 - ось Y

static Matrix reflectionMatrix(int axis);

static Matrix translationMatrix(double dx, double dy);

friend Matrix operator \* (const Matrix &m1, const Matrix &m2);

friend Vector operator \* (const Matrix &m, const Vector &v);

};

#endif // MATRIX\_H

Содержимое файла matrix.cpp:

#include "matrix.h"

#include <QDebug>

Matrix::Matrix(): m{{0, 0, 0}, {0, 0, 0}, {0, 0, 1}}{

}

Matrix::Matrix(const Matrix &newMatrix){

for (int i = 0; i < 3; i++){

for (int j = 0; j < 3; j++){

this->m[i][j] = newMatrix.m[i][j];

}

}

}

Matrix& Matrix::operator = (const Matrix &newMatrix){

for (int i = 0; i < 3; i++){

for (int j = 0; j < 3; j++){

this->m[i][j] = newMatrix.m[i][j];

}

}

return \*this;

}

Matrix operator \* (const Matrix &m1, const Matrix &m2){

Matrix res = Matrix();

for (int i = 0; i < 3; i++){

for (int j = 0; j < 3; j++){

for (int k = 0; k < 3; k++){

res.m[i][j] += m1.m[i][k] \* m2.m[k][j];

}

}

}

return res;

}

Vector operator \* (const Matrix &m, const Vector &v){

Vector res = Vector(0, 0);

res.v[0] = m.m[0][0] \* v.v[0] + m.m[0][1] \* v.v[1] + m.m[0][2] \* v.v[2];

res.v[1] = m.m[1][0] \* v.v[0] + m.m[1][1] \* v.v[1] + m.m[1][2] \* v.v[2];

res.v[2] = m.m[2][0] \* v.v[0] + m.m[2][1] \* v.v[1] + m.m[2][2] \* v.v[2];

return res;

}

Matrix Matrix::rotationMatrix(double angle){

angle \*= M\_PI / 180;

Matrix res;

res.m[0][0] = cos(angle);

res.m[0][1] = -sin(angle);

res.m[1][0] = sin(angle);

res.m[1][1] = cos(angle);

res.m[2][2] = 1;

return res;

}

Matrix Matrix::scalingMatrix(double kx, double ky){

Matrix res;

res.m[0][0] = kx;

res.m[1][1] = ky;

res.m[2][2] = 1;

return res;

}

//0 - ось Х, 1 - ось Y

Matrix Matrix::reflectionMatrix(int axis){

Matrix res = Matrix();

if (!axis){

res.m[0][0] = -1;

res.m[1][1] = 1;

res.m[2][2] = 1;

} else {

res.m[0][0] = 1;

res.m[1][1] = -1;

res.m[2][2] = 1;

return res;

}

}

Matrix Matrix::translationMatrix(double dx, double dy){

Matrix res = Matrix();

res.m[0][0] = 1;

res.m[1][1] = 1;

res.m[2][2] = 1;

res.m[0][2] = dx;

res.m[1][2] = dy;

return res;

}

Содержимое файла vector.h:

#ifndef VECTOR\_H

#define VECTOR\_H

#include <QPointF>

#include <iostream>

class **Matrix**;

class **Vector**{

public:

double v[3];

**Vector**(double x, double y);

**Vector** (QPointF &p);

**Vector** (){v[0] = 0; v[1] = 0; v[2] = 1;}

double **x**();

double **y**();

double operator[](unsigned i);

QPoint **getPoint**();

void **setPoint**(QPoint);

friend Vector operator \* (const Matrix &m, const Vector &v);

};

#endif // VECTOR\_H

Содержимое файла vector.cpp:

#include "vector.h"

Vector::Vector(double x, double y){

v[0] = x;

v[1] = y;

v[2] = 1;

}

Vector::Vector(QPointF &p){

v[0] = p.x();

v[1] = p.y();

v[2] = 1;

}

double Vector::x(){

return v[0];

}

double Vector::y(){

return v[1];

}

double Vector::operator[](unsigned int i){

return v[i];

}

QPoint Vector::getPoint(){

return QPoint(v[0], v[1]);

}

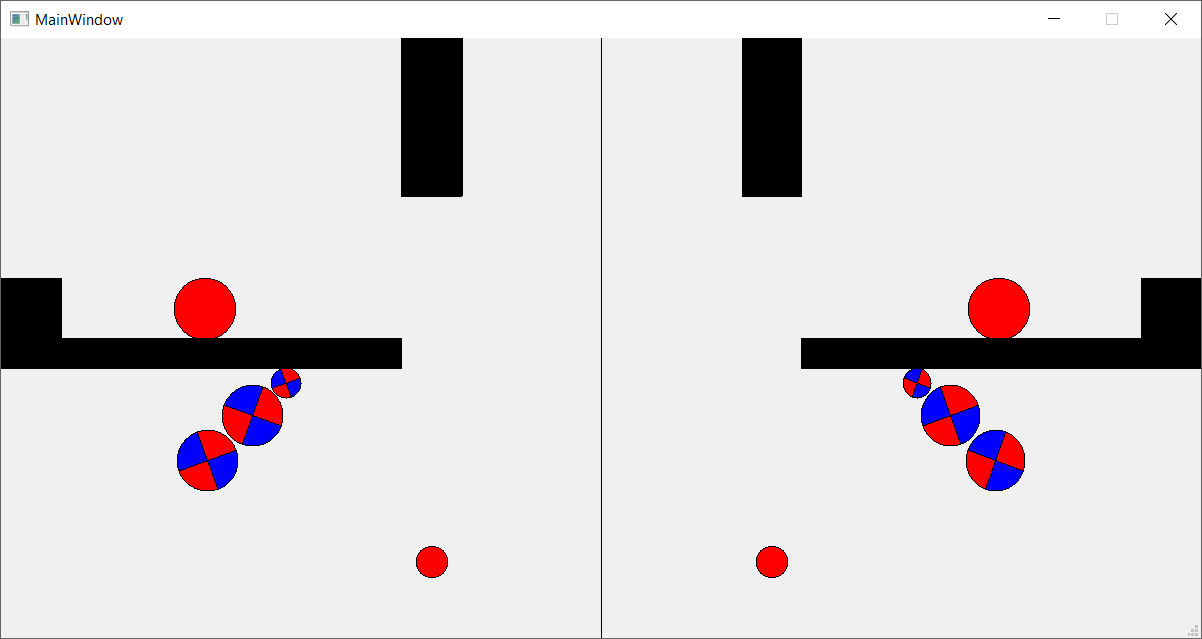
void Vector::setPoint(QPoint x){

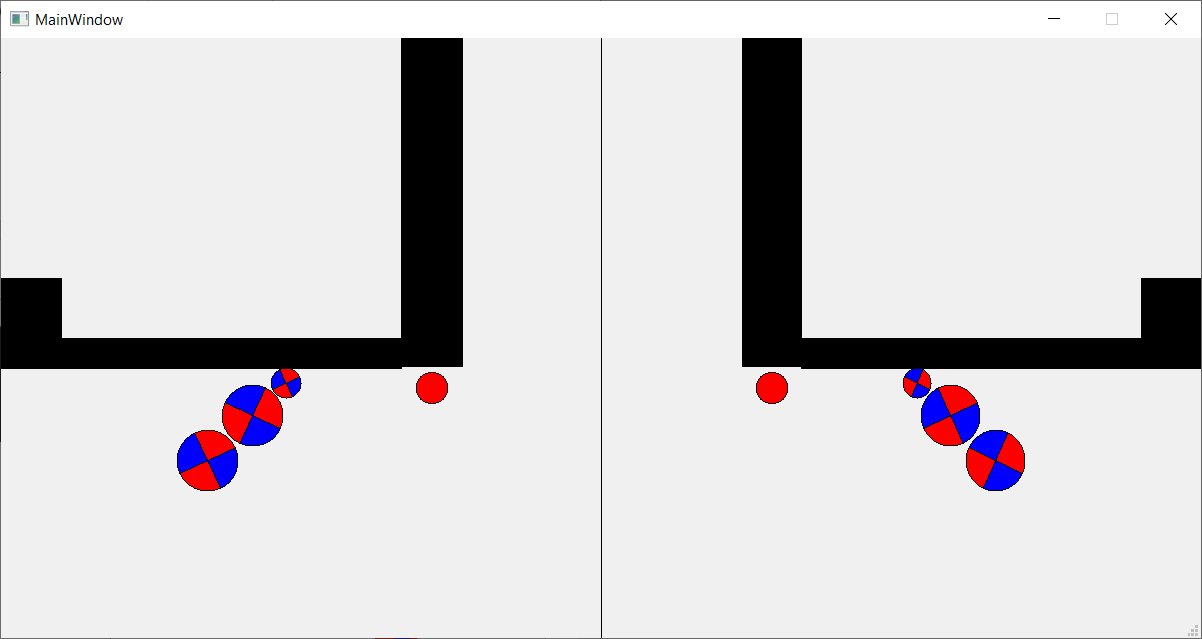
v[0] = x.x();

v[1] = x.y();

v[2] = 1;

}



z