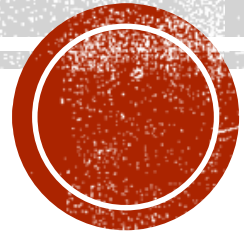
Нижегородский государственный университет им. Н.И. Лобачевского

Центр дополнительного профессионального образования

Программа дополнительного профессионального образования «Профессиональное программирование»

ВЫПУСКНАЯ РАБОТА

«МОДЕЛИРОВАНИЕ ИГРЫ БИЛЬЯРД «ВОСЬМЕРКА» В СРЕДЕ QT CREATOR»

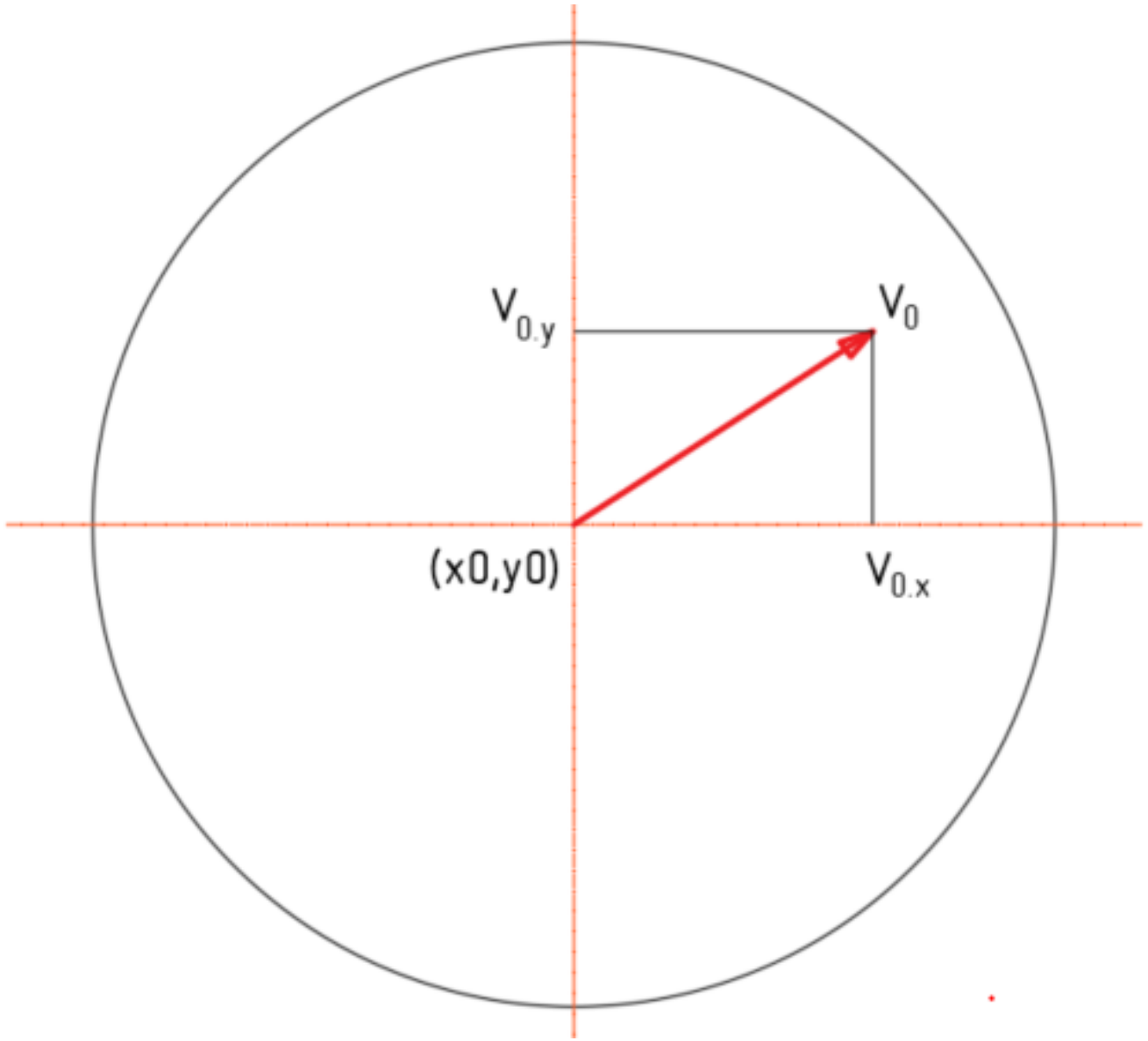


Выполнил: Родин Е.В.

Руководитель: Городецкий С.Ю.

Нижний Новгород, 2021г

ОБЪЕКТ КЛАСС BALL



Balls	<16 items>	QList<Ball *>
*([0])	@0x228830b06d0	Ball
[QObject]	@0x228830b06d0	QObject
[QGraphicsItem]	@0x228830b06e0	QGraphicsItem
[Point]	@0x228830b0728	Point
[vptr]	__vfptr	
t	0.02	double
v0	@0x228830b0748	V
x	-11.977212597826068	double
y	-2719.973629721139	double
x0	148.17751879699247	double
y0	355.21804511278197	double
[d]	@0x228830aebb0	QObjectPrivate
[parent]	0x0	QObject *
[children]	<0 items>	QList<QObject *>
[properties]	<at least 0 items>	
[methods]	<0 items>	
[extra]		
*_font	@0x2288514b650	QFont
active	true	bool
calculated	false	bool
color	@0x228830b06fc	QColor
moving	true	bool
number	0	int
r	11.421052631578947	double
scale	0.23308270676691728	double
staticMetaObj...	@0x7ff6cae4aab0	QMetaObject
type	WHITE (3)	Ball::Type

МОДЕЛЬ ДВИЖЕНИЯ ШАРА

В качестве физической модели движения шара использовано уравнение движение, согласно которому за каждый промежуток deltaT координата меняется на V0.x*deltaT и V0.y*delatT при этом:

V0*=deltaV, а **deltaV*=deltaForce**. В

Параметры изменения значение скоростей и ускорений подбирались опытным путем.

Движение шара происходит до момента, пока модуль изменения скорости по обеим координатам не станет меньше значения **deltaVmin**. После чего шару присваивается статус неподвижного. Это сделано для остановки шара и предотвращения его движения в обратном направлении, что неминуемо бы произошла из использования полинома в качестве функции, описывающей движение.

x=x+V0.x*deltaT;

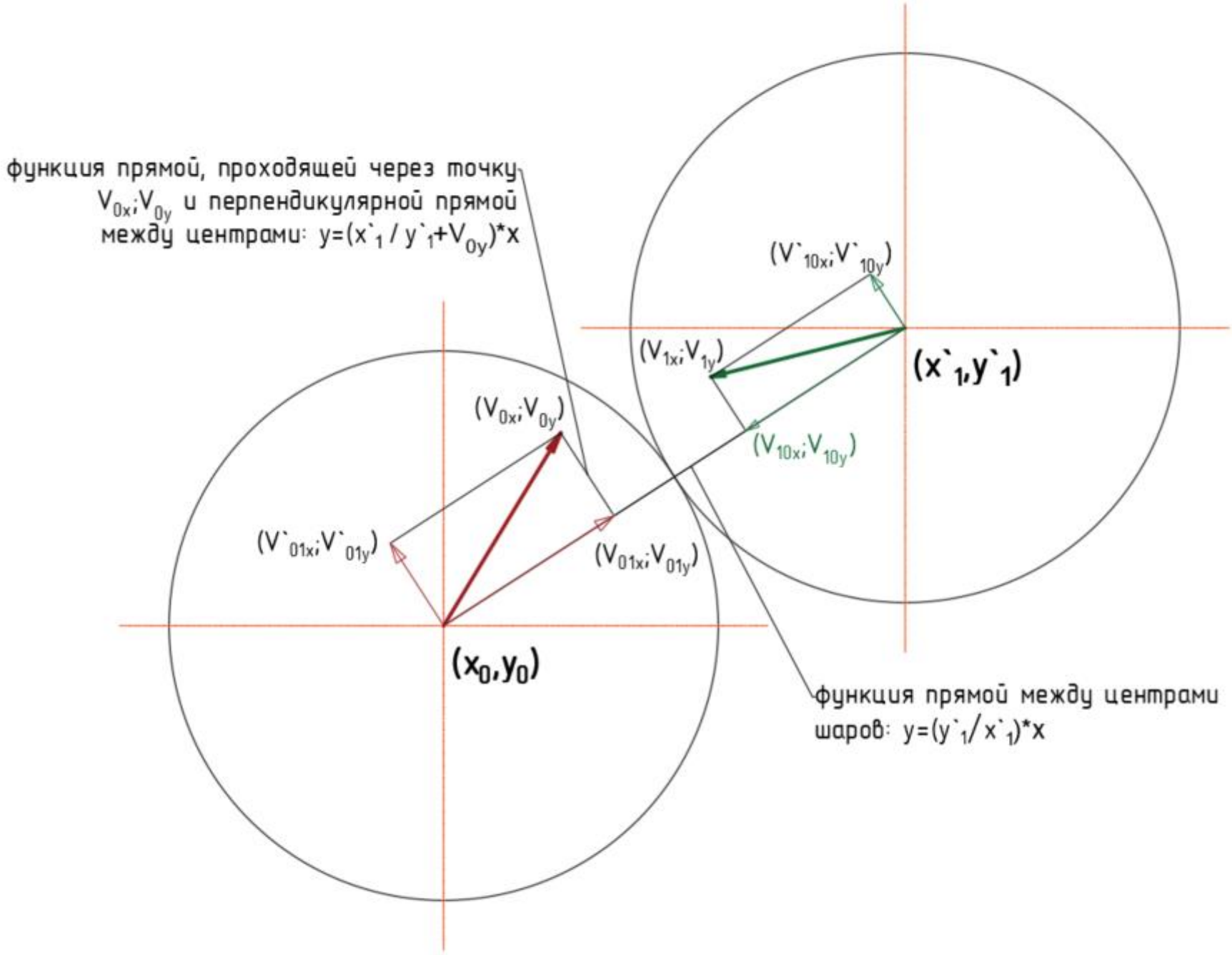
y=y+V0.y*deltaT;

V0*=deltaV;

deltaV*=deltaForce;

```
void Ball::move(){
    moveBy(this->v0.x * MainWindow::deltaT, this->v0.y * MainWindow::deltaT);
    V vPrev = this->v0;
    this->v0 *= MainWindow::deltaV;
    MainWindow::deltaV*=MainWindow::deltaForce;
    if ((fabs(vPrev.x - this->v0.x) < MainWindow::deltaVmin) &&
            (fabs(vPrev.y - this->v0.y) < MainWindow::deltaVmin)){
        this->setMovingStatus(false); }}
```

СОУДАРЕНИЕ ДВУХ ШАРОВ



Уравнение прямой $y = \frac{y`_1}{x`_1} * x$

Ищем уравнение прямой, перпендикулярной к данной, проходящей через точку:

$$y - y_1 = \frac{1}{k} \cdot (x - x_1)$$

Подставляем значение из функции:

$$y = \left(\frac{x`_1}{y`_1} + V_{0y}\right) \cdot x$$

Координаты проекции вектора скорости первого шара на прямую между центрами:

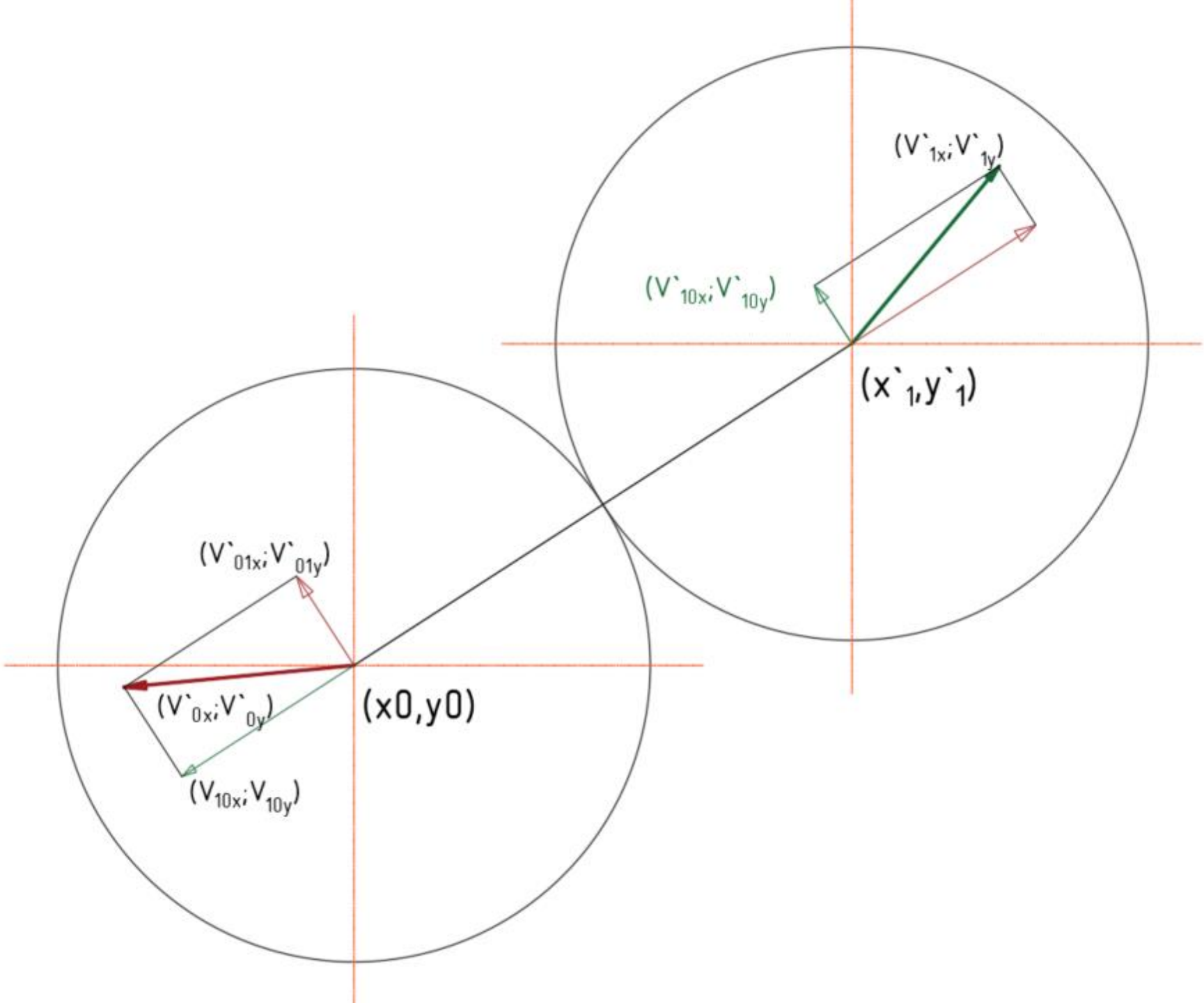
$$V_{01x} = \frac{\left(\frac{x`_1}{y`_1} * V_{0x} + V_{0y}\right)}{\left(\frac{x`_1}{y`_1} + \frac{y`_1}{x`_1}\right)}$$

$$V_{01y} = \frac{y`_1}{x`_1} * V_{01x}$$

Получив V_{01} мы находим перпендикулярный ему вектор:

$V`_{01} = V_0 - V_{01}$

СОУДАРЕНИЕ ДВУХ ШАРОВ. ПРОДОЛЖЕНИЕ

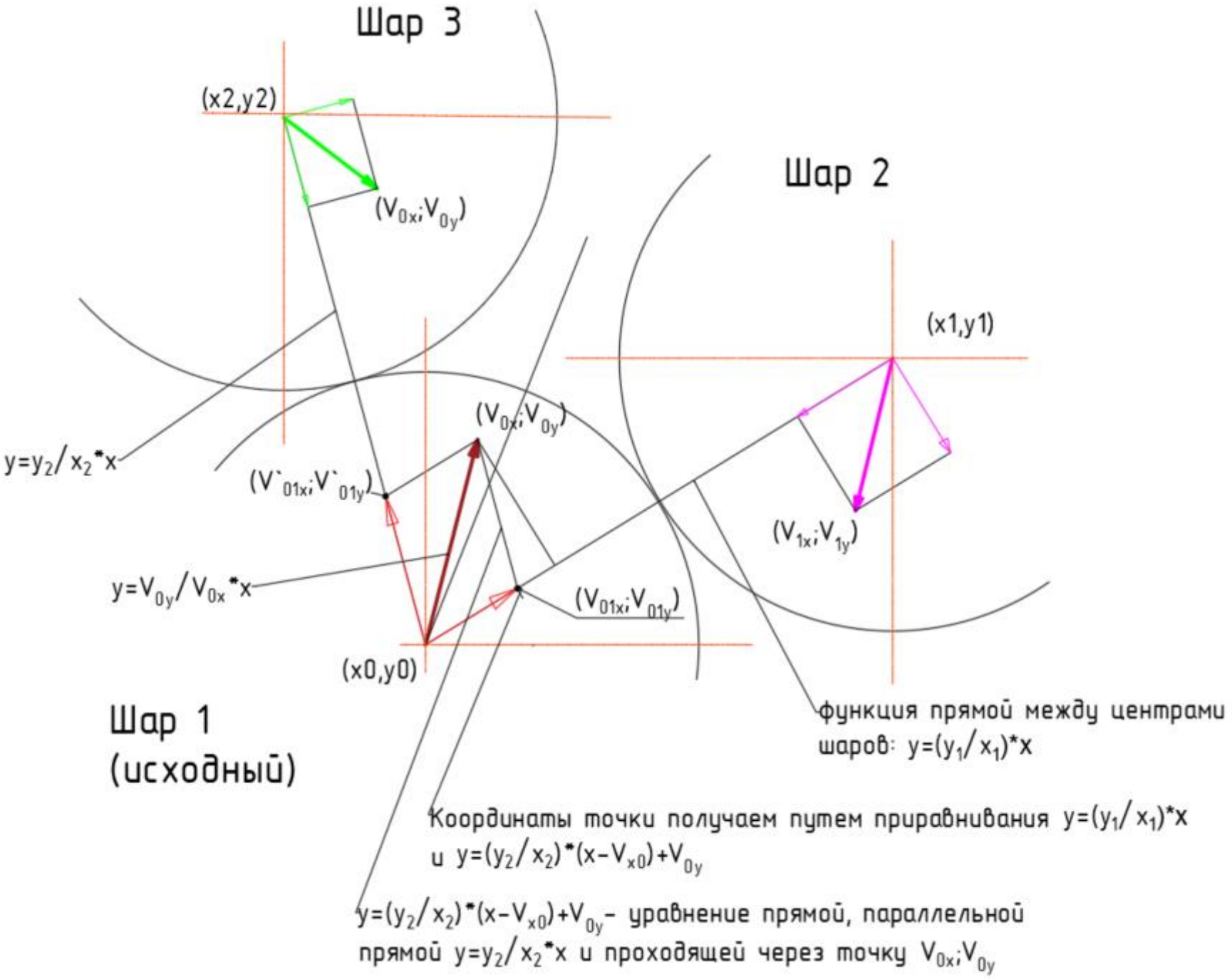


Аналогичные действия проводим для второго шара и ищем у него векторы V_{10} и $V`_{10}$

Обмениваем векторы V_{01} и V_{10} между собой.

Собираем векторы скорости для каждого шара. Итог на рисунке.

МОДЕЛЬ СОУДАРЕНИЯ ТРЕХ ШАРОВ



Получаем уравнение прямых:

$$y = \frac{y_1}{x_1} * x \quad \text{и} \quad y = \frac{y_2}{x_2} * x$$

Далее ищем прямую, параллельную $y = \frac{y_2}{x_2} * x$ и проходящую через точку $(V_{0x};V_{0y})$:

$$y = \frac{y_2}{x_2}(x - V_{x0}) + V_{0y}$$

Находим координаты вектора:

$$V_{01x} = \frac{\left(V_{0y} - \frac{y_2}{x_2} * V_{0x}\right)}{\left(\frac{y_1}{x_1} - \frac{y_2}{x_{21}}\right)}$$

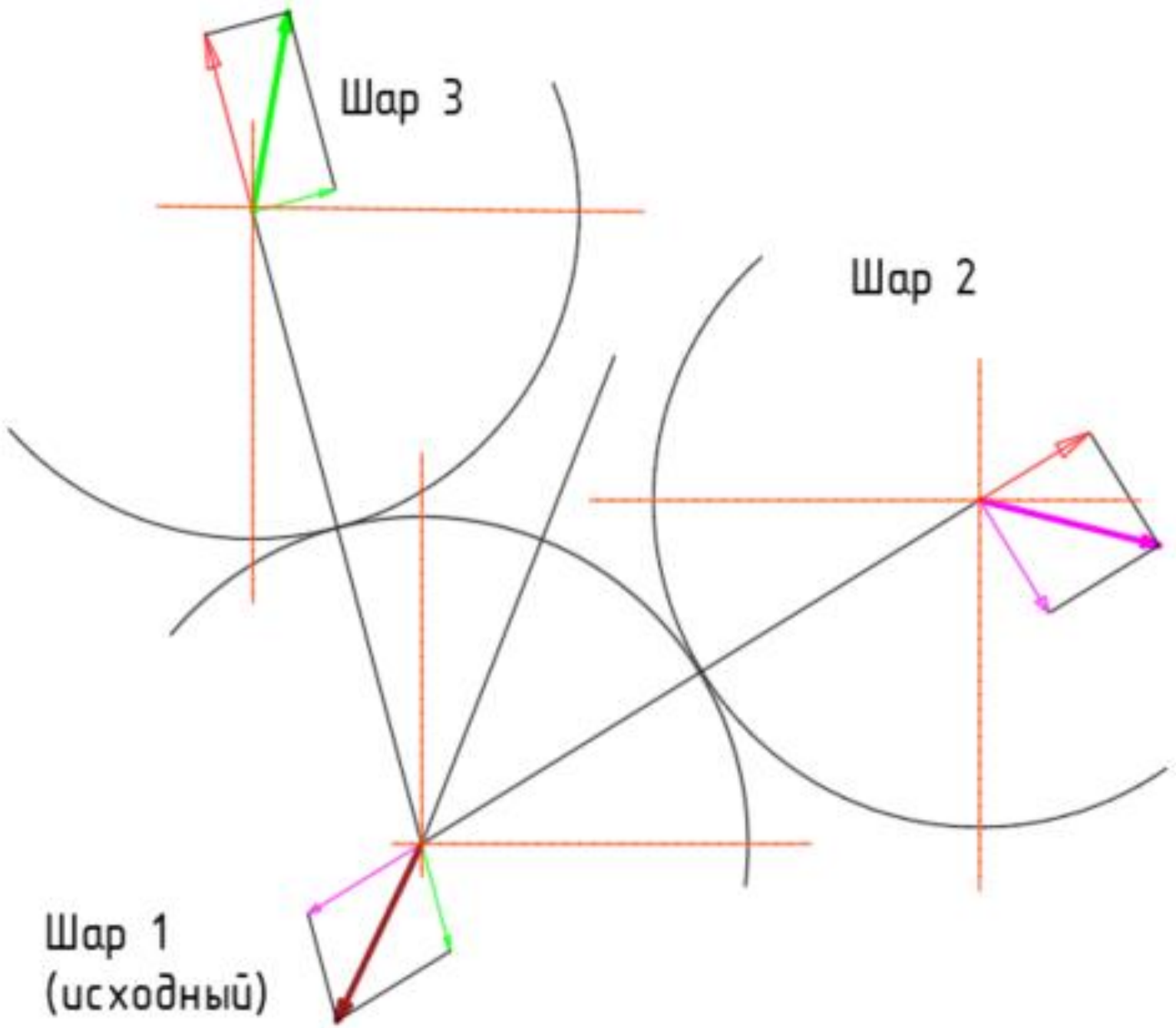
$$V_{01y} = \frac{y_1}{x_1} * V_{01x}$$

Получив V_{01} мы находим перпендикулярный ему вектор:

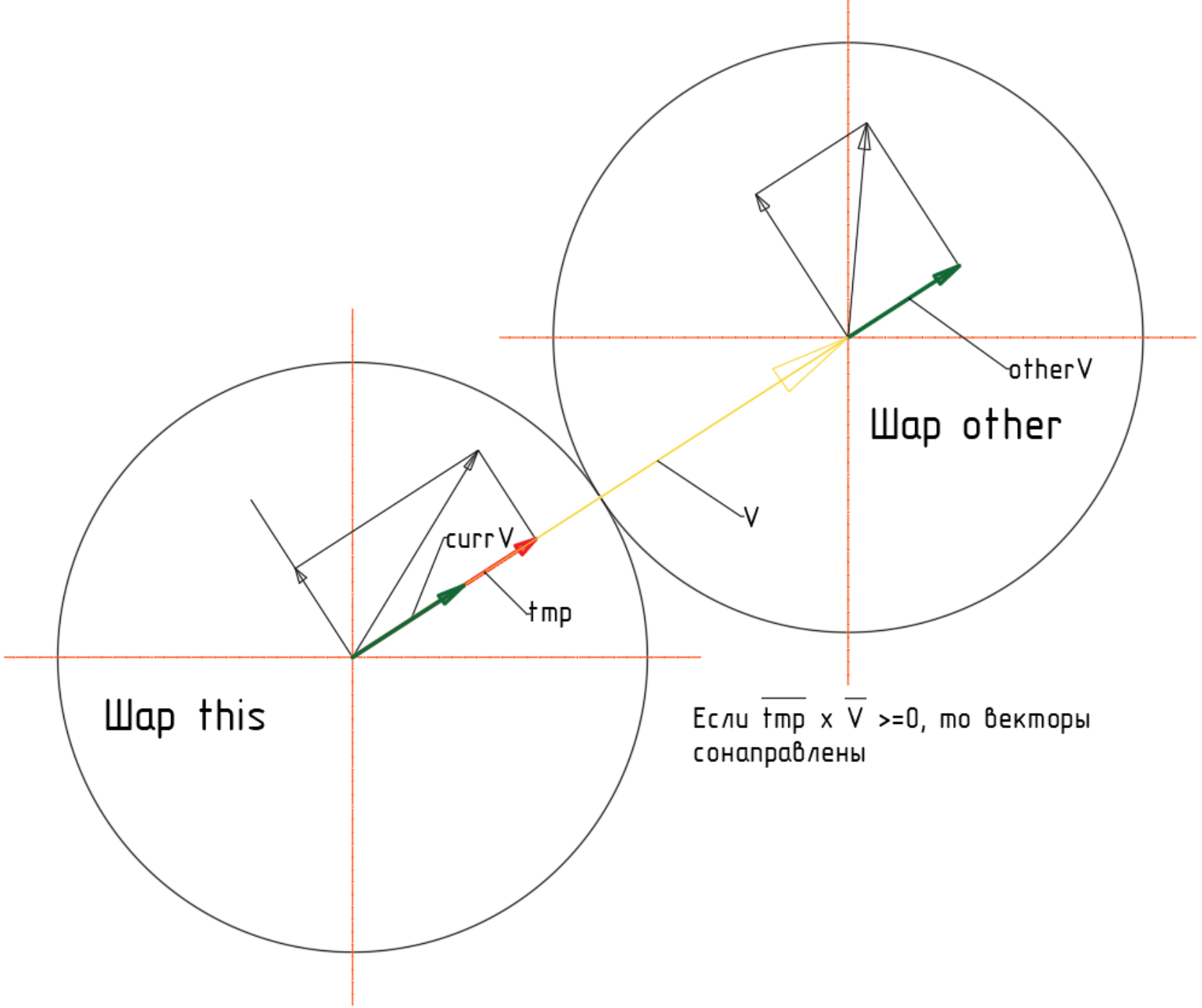
$$V`_{01} = V_0 - V_{01}$$

МОДЕЛЬ СОУДАРЕНИЯ ТРЕХ ШАРОВ. ПРОДОЛЖЕНИЕ

После сложение векторов и общего вектора скорости, имеем следующую картину



СЛОЖНОСТИ РЕАЛИЗАЦИИ



При вызове метода

void MainWindow::calcBallsDistances()

для расчета дистанции между шарами и последующего вызова функции соударения шаров может наступить момент, когда шары столкнулись на предыдущей итерации, но не успели «отскочить» на достаточное расстояние друг от друга из-за потери скорости и для них вновь будет вызвана функция соударения и шара войдут в зацикленное биение друг о друга.

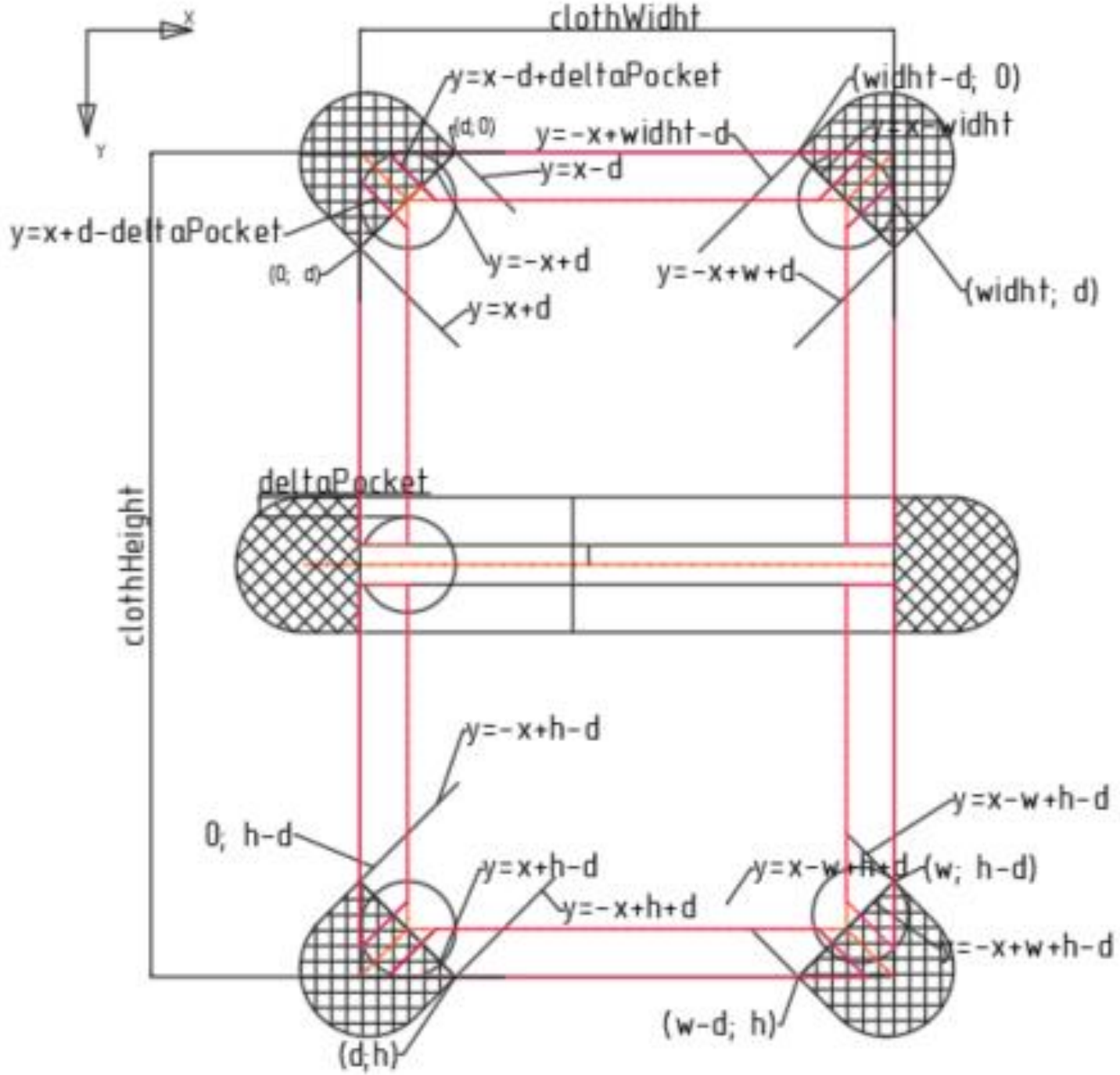
Суть метода сводится к вычислению скалярного произведения между вектором направленным от центра текущего шара к центру противоположного – **V** и разницей проекций векторов скорости шаров на межцентровую прямую

tmp=currV-otherV.

При **tmp x V>=0** векторы

сонаправлены

МОДЕЛЬ СТОЛА



Стол описан множеством прямых, при пересечении который для шара вызываются соответствующие события:

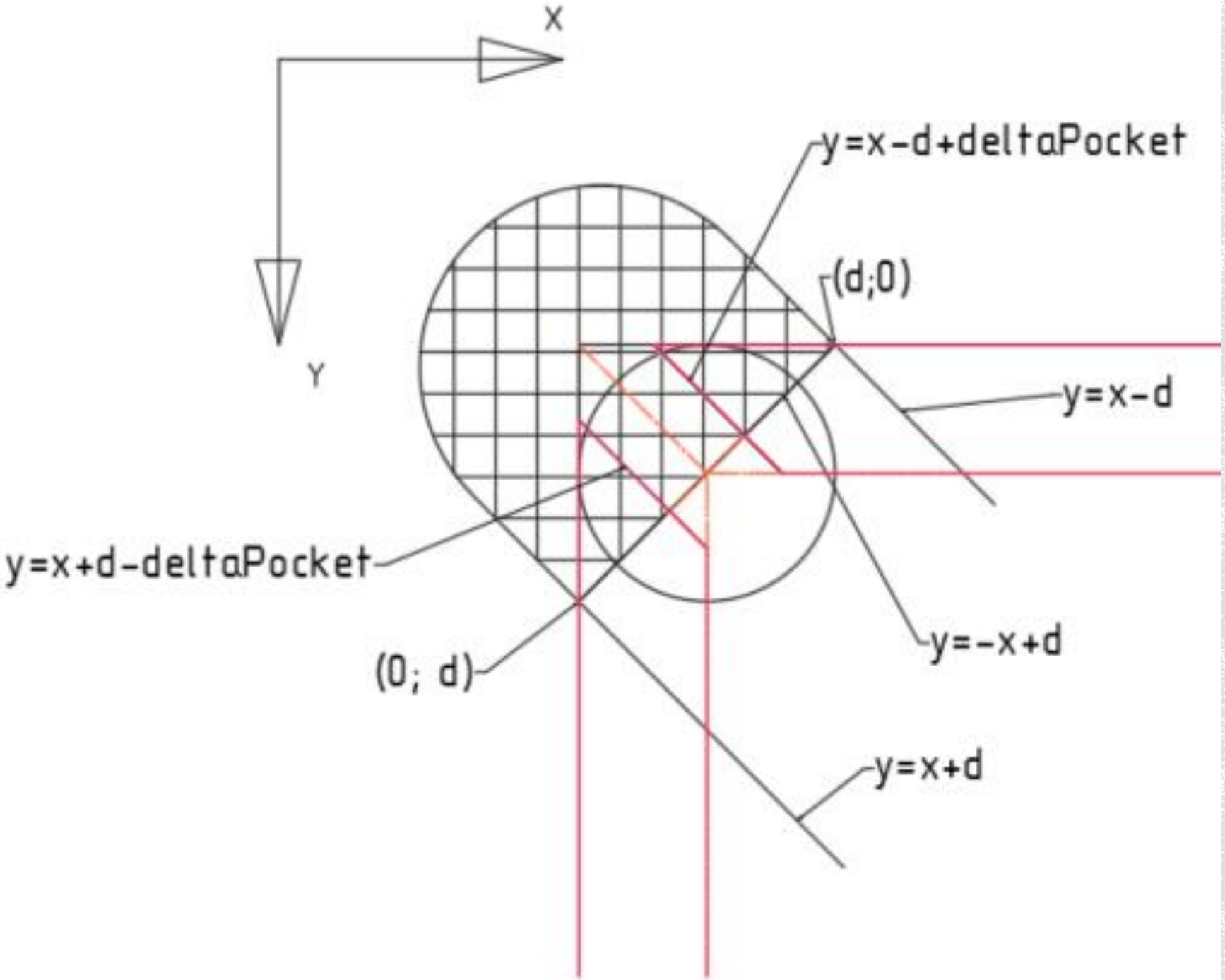
-отражение от борта;

-попадание в лузу.

Прямые описаны уравнениями, коэффициенты и слагаемые которых определяются в зависимости от размера шара и размеры лузы. Данные параметры автоматически пересчитывают при изменении размеров шаров и луз, зависящих от размера стола.

Уравнения соответствующих бортов и луз показаны на рисунке.

ПОПАДАНИЕ ШАРА В ЛУЗУ



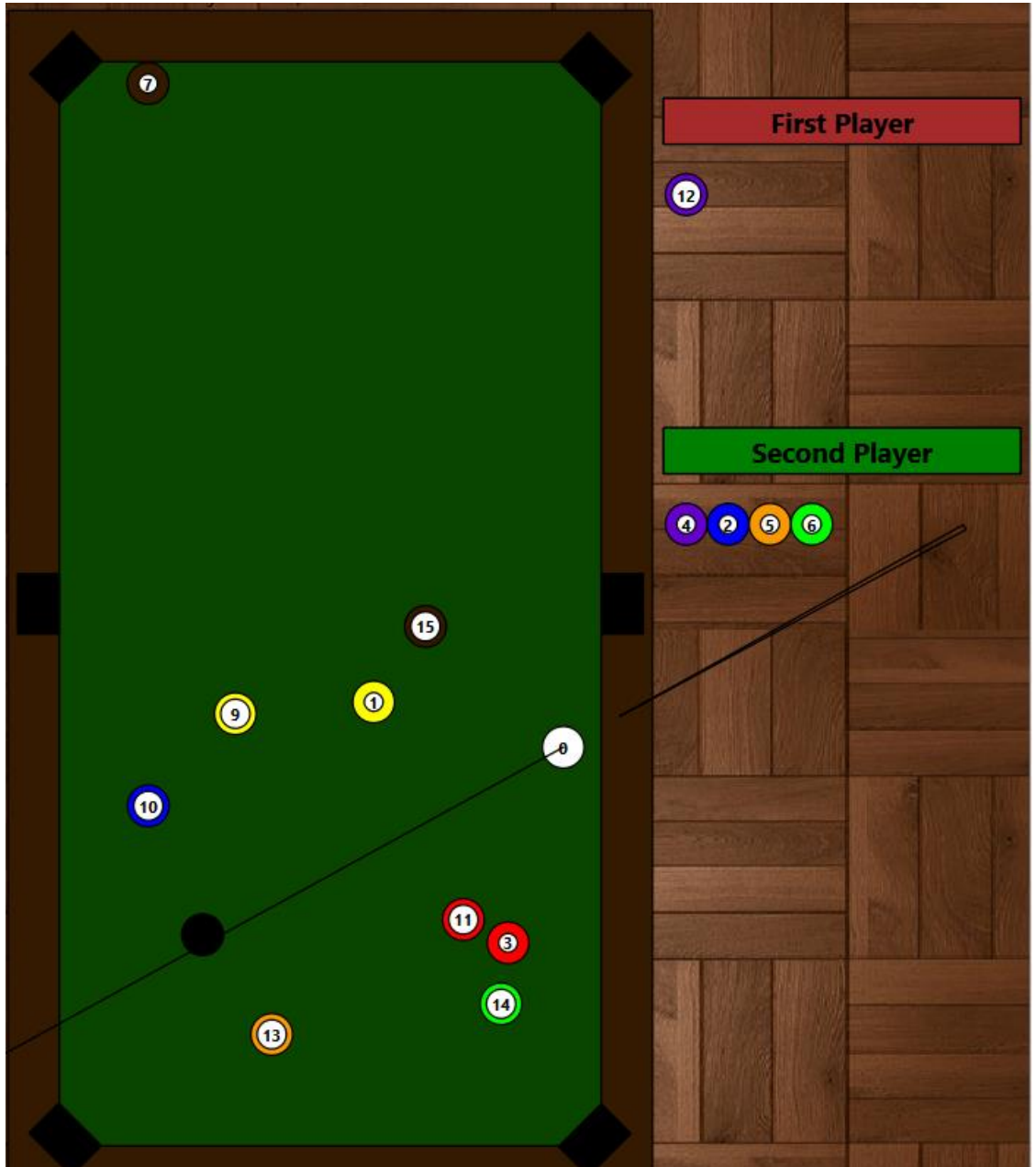
Зоны луз задаются аналитически с помощью уравнений прямых в координатах стола. Пример функций для верхней левой лузы показан на рисунке.

За реализацию в программе отвечает метод

void MainWindow::**pocketChech**();

Помимо проверки попадания в лузу, метод так же определяет тип забитого шара и присваивает его соответствующему игроку или выставляет статус «Фол»

ОБЩИЙ ИНТЕРФЕЙС



На сцене кроме стола с объектами классов **Ball** и **Cue**

Находятся два объекта класса **Player.**

У каждого из этих объектов есть своя область размещения забиты шаров.

Активный игром помечен зеленым цветом

В момент подготовки к удары планка игрока закрашивается соразмерно с силой удара.

РАСЧЕТ КАДРА

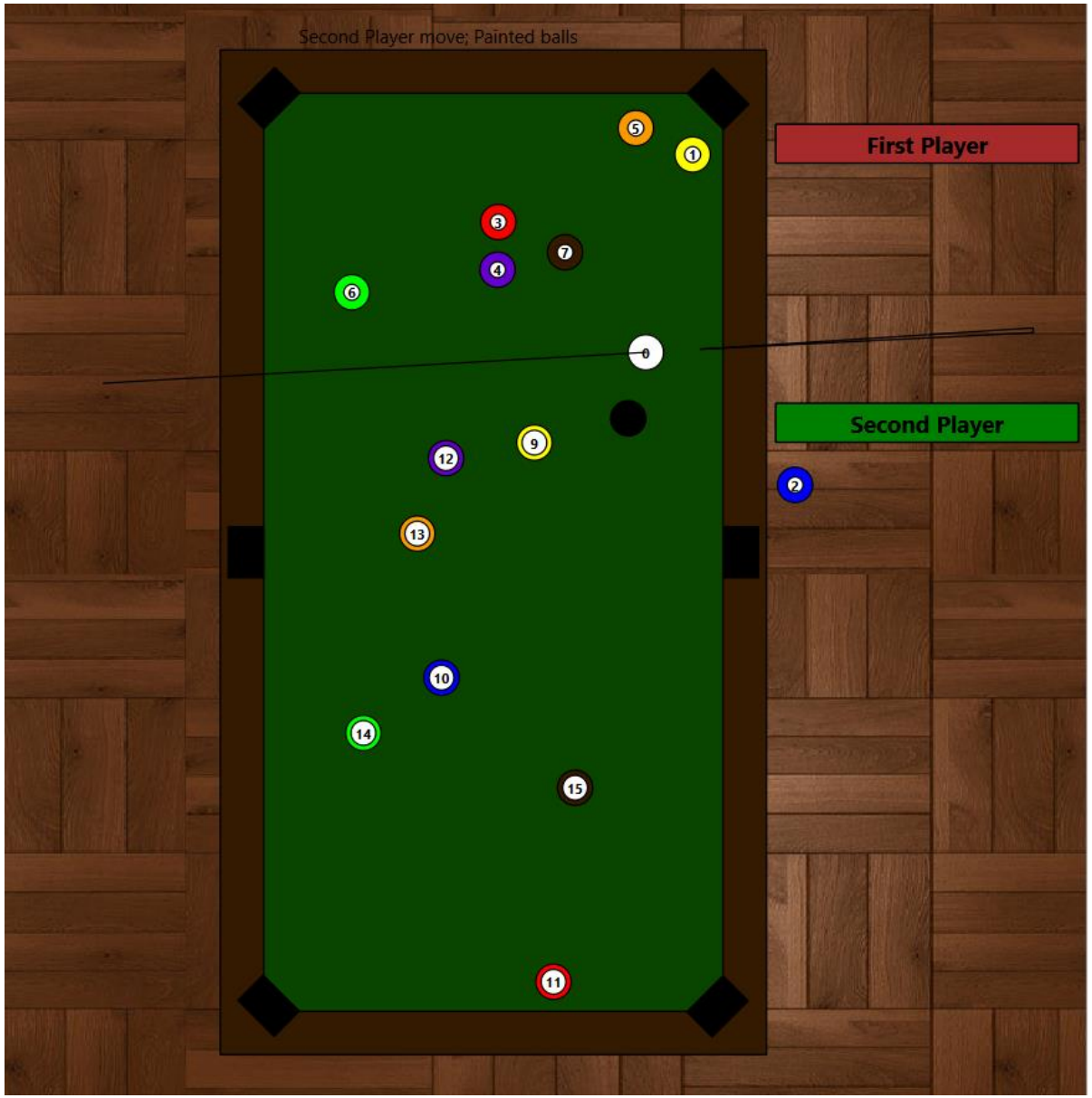
За расчет кадра отвечает метод класса MainWindow void MainWindow::**pitch**(), связанный с таймером QTimer * timer

Реализация:

```
void MainWindow::pitch(){
    for(int i =0;i<40;++i)   {
        setCalculetedStatusFalse();    //выставляем статус "просчитан" в false для всех шаров
        pocketChech();                          //проверяем падение в лузу
        boardCollision();                       //проверяем столкновение с бортом
        calcBallsDistances();                   //проверяем дистанцию между шарами
        moveBalls();    //продвигаем все находящиеся в движении шары на одну итерацию
    }
    scene->update();                    //отрисовываем сцену
    if (!ballsInMoving()) {                         //Обработка завершения движения шаров
timer->stop();                                              //останавливаем таймер
        checkPocketedBalls();                               //проверяем забитые шары
        setCuePosition();                           //устанавливаем кий на белый шар
    }
}
```

- На каждый тик таймера (60 раз в секунду) происходит дополнительно определенное количество итераций за счет использования цикла for. Это необходимо для обеспечения нужной точности движения и экономит ресурсы системы на излишнюю отрисовку сцены.
- В каждом цикле вызова метода **pitch()** проверяется падение шара в лузу, столкновение с бортом, высчитывается дистанция для всех шаров и происходит продвижение всех шаров на один шаг deltaT.
- Далее проверяет двигается ли хотя бы один шар на столе, если движение прекращается, таймер останавливается и запускается метод проверки упавших шаров в лузу **checkPocketedBalls()** в котором выставляются соответствующие флаги, влияющие на дальнейшее течение игры (фол, продолжение хода и завершение игры).

МЕТОД РАСЧЕТА РАССТОЯНИЙ МЕЖДУ ШАРАМИ И ВЫЯВЛЕНИЕ СТОЛКНОВЕНИЙ



В методе последовательно проверяются все движущиеся активные шары на квадрат дистанции между другими шара. При уменьшении этого параметра меньше квадрата диаметра шаров, вызывается метод столкновения двух или трех шаров в зависимости от того, сколько шаров находится в диапазоне досягаемости шара. При просчете шара ему присваивается статус **“calculated”**, чтобы не просчитывать его внутри вложенного цикла.

ПРЕИМУЩЕСТВА И НЕДОСТАТКИ. ПУТИ РАЗВИТИЯ

Преимущества:

- Полностью удалось избежать использование тригонометрических функций **s**in() и cos().
- При для оценки расстояния между шарами значение дистанции заменено на ее квадрат.

Недостатки:

- Упрощенная математическая модель движения шаров. Отсутствует вращение шара.
- Упрощенный отскок шаров от бортов стола в области луз

Пути развития:

- Возможность удара кием не только в центр шара, но и с некоторым смещением.
- Многопользовательский режим
- Пересмотр реализации класс; отказ от статических переменных.
- Изменить механизм наследования