JĘZYKI ASEMBLEROWE PROJEKT „SYMULATOR ZDERZEŃ”

KRZYSZTOF DEC

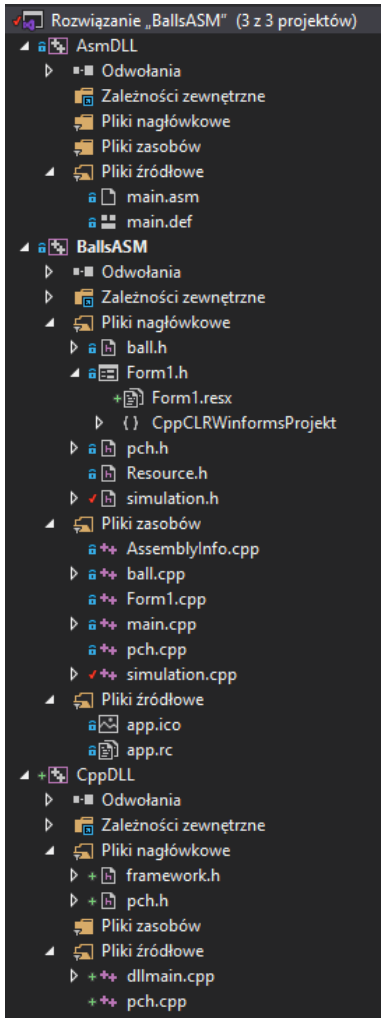
GRUPA 1, SEKCJA 2

INFORMATYKA AEI SEM.5

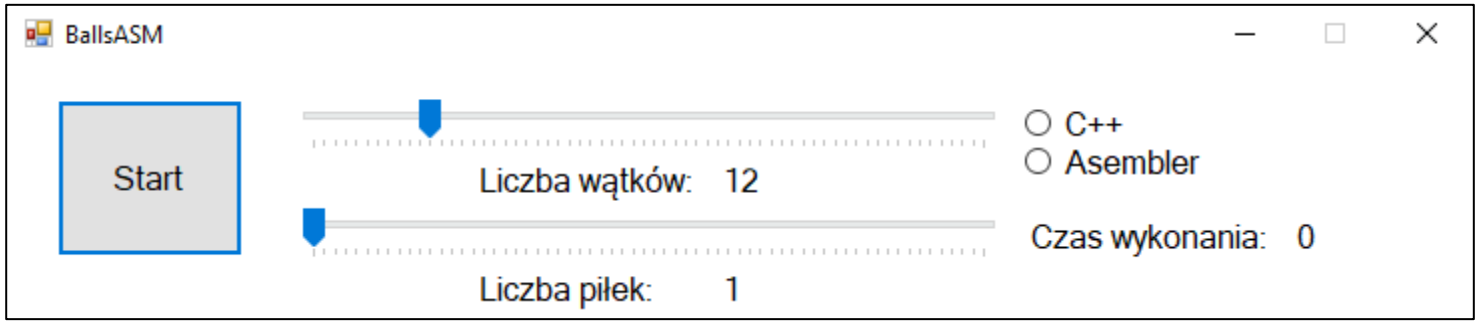
ANALIZA TEMATU

Składowe projektu:

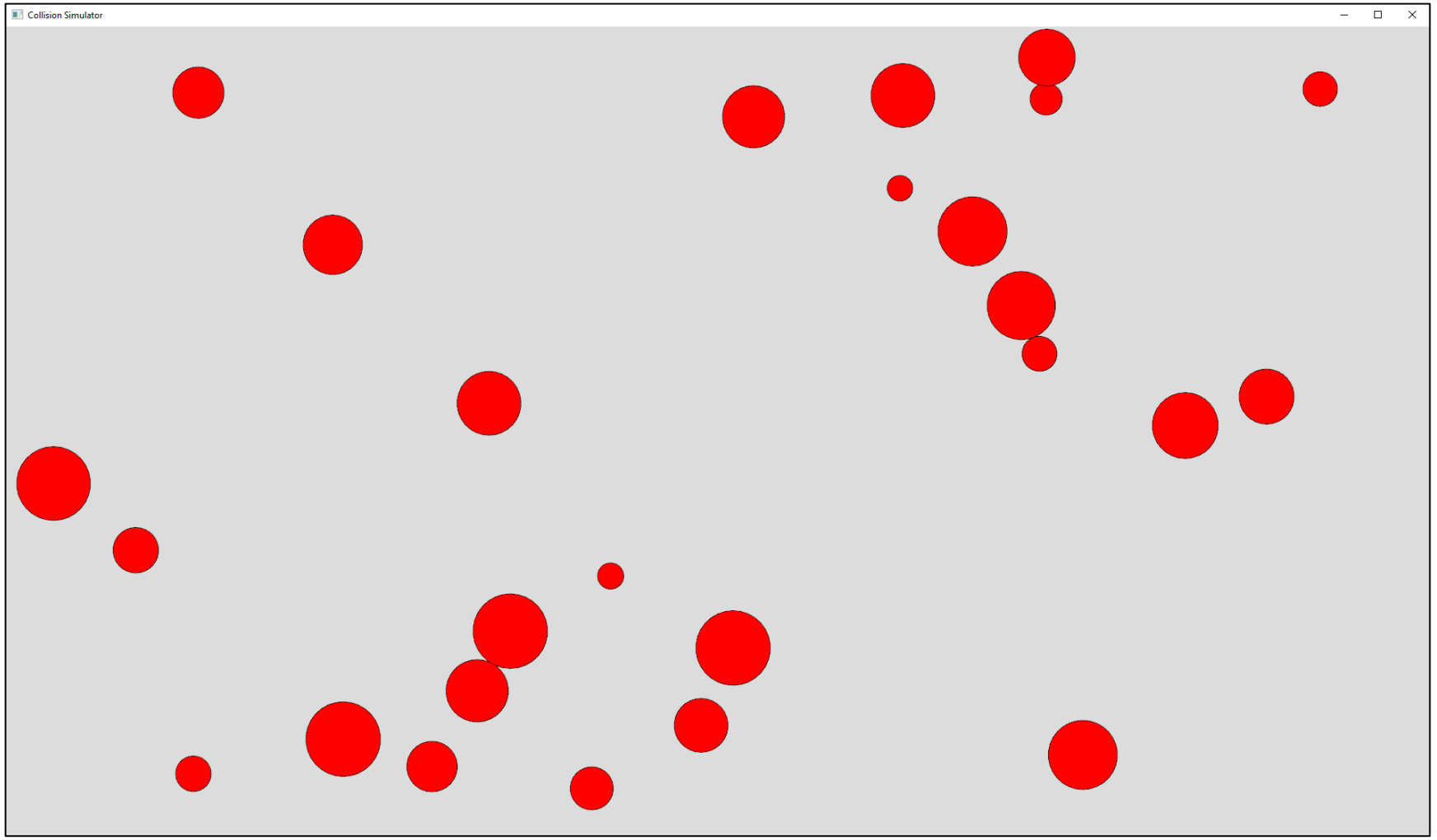
- Program główny z graficznym interfejsem użytkownika (SFML)
  - Komunikacja z użytkownikiem
  - Wybranie parametrów wykonania programu
- Biblioteka DLL w asemblerze
- Biblioteka DLL w języku C++



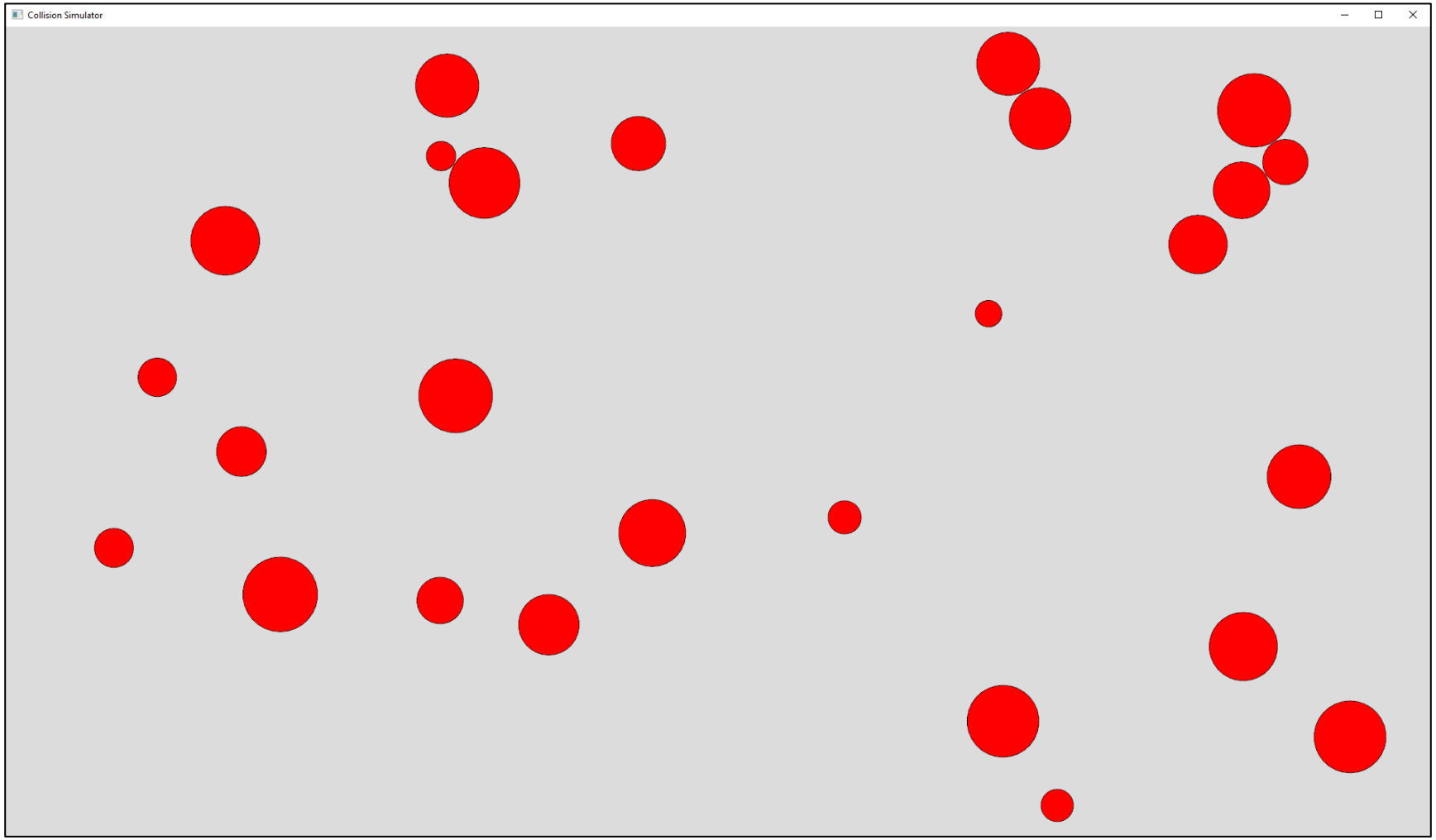
INTERFEJS UŻYTKOWNIKA



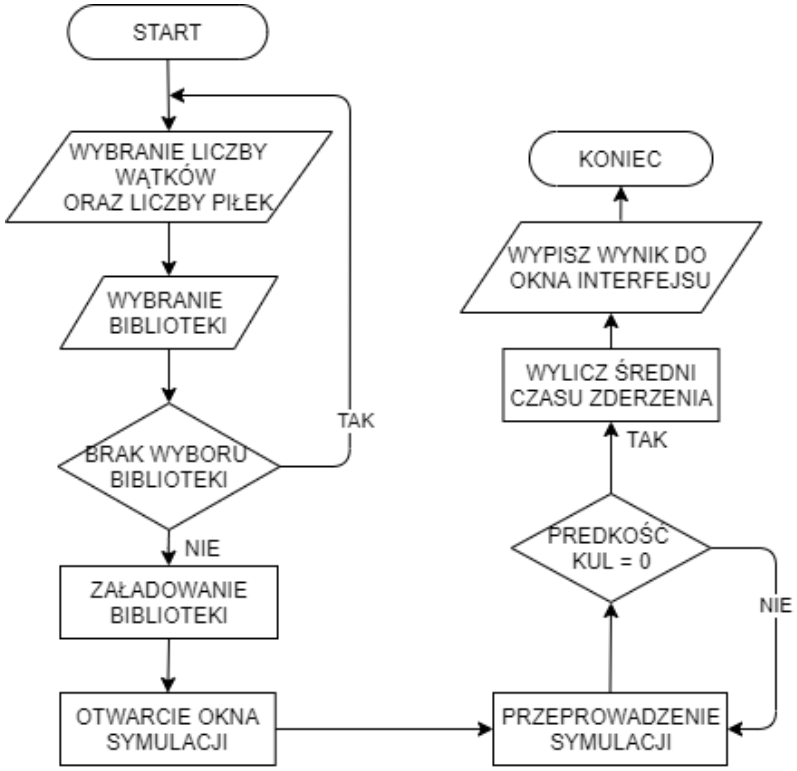
INTERFEJS UŻYTKOWNIKA – CD.



WYNIK DZIAŁANIA PROGRAMU



SCHEMAT BLOKOWY PROGRAMU



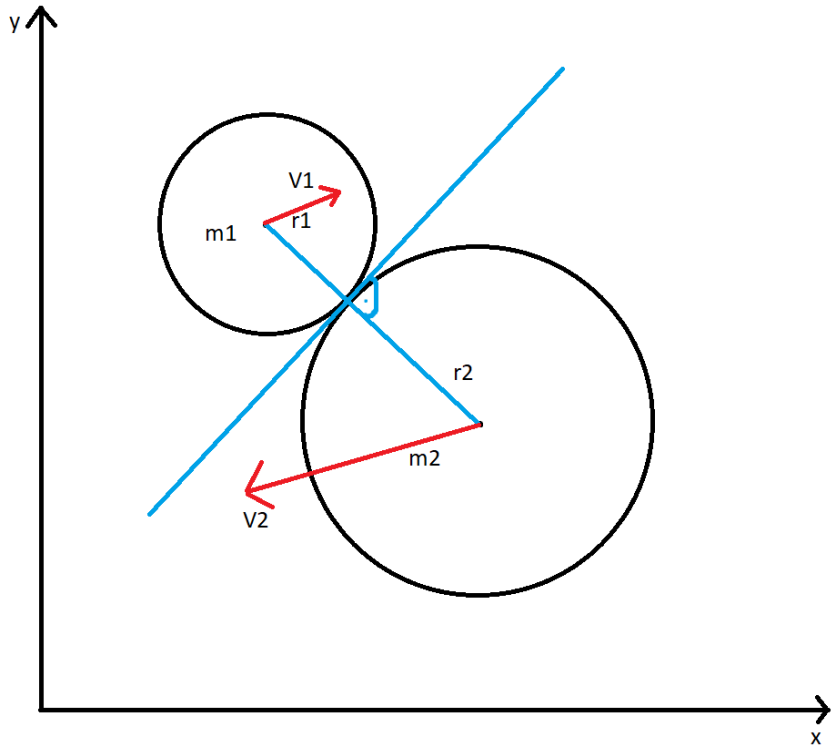
TEORIA PROJEKTU

Poruszane problemy w projekcie:

- Wykrywanie zderzeń obiektów
- Dwuwymiarowa zasada zachowania pędu
- Odbicia nieliniowe sprężyste

ODBICIE W TEORII

ODBICIE W PRAKTYCE



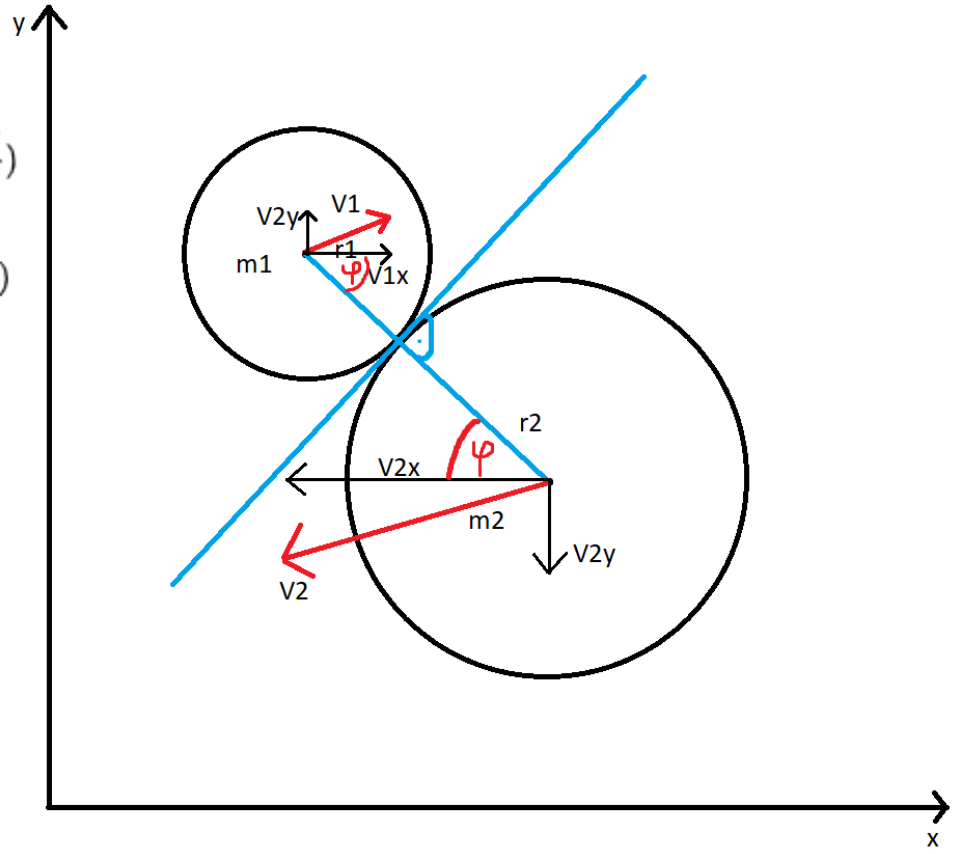
ODBICIE W PRAKTYCE

$$v'_{1x} = \frac{v_1 \cos(\theta_1 - \varphi)(m_1 - m_2) + 2m_2 v_2 \cos(\theta_2 - \varphi)}{m_1 + m_2} \cos(\varphi) + v_1 \sin(\theta_1 - \varphi) \cos(\varphi + \frac{\pi}{2})$$

$$v'_{1y} = \frac{v_1 \cos(\theta_1 - \varphi)(m_1 - m_2) + 2m_2 v_2 \cos(\theta_2 - \varphi)}{m_1 + m_2} \sin(\varphi) + v_1 \sin(\theta_1 - \varphi) \sin(\varphi + \frac{\pi}{2})$$

$$\mathbf{v}'_1 = \mathbf{v}_1 - \frac{2m_2}{m_1 + m_2} \frac{\langle \mathbf{v}_1 - \mathbf{v}_2, \mathbf{x}_1 - \mathbf{x}_2 \rangle}{\|\mathbf{x}_1 - \mathbf{x}_2\|^2} (\mathbf{x}_1 - \mathbf{x}_2),$$

$$\mathbf{v}'_2 = \mathbf{v}_2 - \frac{2m_1}{m_1 + m_2} \frac{\langle \mathbf{v}_2 - \mathbf{v}_1, \mathbf{x}_2 - \mathbf{x}_1 \rangle}{\|\mathbf{x}_2 - \mathbf{x}_1\|^2} (\mathbf{x}_2 - \mathbf{x}_1)$$



ODBICIE W KODZIE PROGRAMU

?

```
//static collision
for (auto& ball : vecBalls) {
    for (auto& target : vecBalls) {
        if (ball.getId() != target.getId()) {
            if (doBallsOverlap(ball, target)) {
                vecCollisions.push_back(std::make_pair(ball, target));
                float distance = distanceBetweenBalls(ball, target);
                float overlap = 0.5f * (distance - ball.getRadius() - target.getRadius());
                ball.getBall().move(-overlap * (ball.getPosX() - target.getPosX()) / distance, -overlap * (ball.getPosY() - target.getPosY()) / distance);
                target.getBall().move(overlap * (ball.getPosX() - target.getPosX()) / distance, overlap * (ball.getPosY() - target.getPosY()) / distance);
            }
        }
    }
    if (ball.getPosX() - ball.getRadius() < 0) {
        ball.getBall().move(ball.getRadius() - ball.getPosX(), 0);
    }
    if (ball.getPosX() + ball.getRadius() > WindowLenght) {
        ball.getBall().move(WindowLenght - ball.getPosX() - ball.getRadius(), 0);
    }
    if (ball.getPosY() - ball.getRadius() < 0) {
        ball.getBall().move(0, ball.getRadius() - ball.getPosY());
    }
    if (ball.getPosY() + ball.getRadius() > WindowHeight) {
        ball.getBall().move(0, WindowHeight - ball.getPosY() - ball.getRadius());
    }
}

//dynamic collision
for (auto collision : vecCollisions) {
    //thread handling here
    Ball ball1 = collision.first;
    Ball ball2 = collision.second;
}
```

ODBICIE W KODZIE PROGRAMU

```
//1
float distance = distanceBetweenBalls(ball1, ball2);

float normalx = (ball2.getPosX() - ball1.getPosY()) / distance;
float normaly = (ball2.getPosY() - ball1.getPosY()) / distance;

float kX = ball1.getSpeedX() - ball2.getSpeedX();
float kY = ball1.getSpeedY() - ball2.getSpeedY();

float p = 2.0 * (normalx * kX + normaly * kY) / (ball1.getRadius() + ball2.getRadius());

speedX = ball1.getSpeedX();
speedY = ball1.getSpeedY();
ball1.setSpeedX(speedX - p * ball2.getRadius() * normalx);
ball1.setSpeedY(speedY - p * ball2.getRadius() * normaly);
speedX = ball2.getSpeedX();
speedY = ball2.getSpeedY();
ball2.setSpeedX(speedX + p * ball1.getRadius() * normalx);
ball2.setSpeedY(speedY + p * ball1.getRadius() * normaly);

//2
distance = distanceBetweenBalls(ball1, ball2);
normalx = (ball2.getPosX() - ball1.getPosY()) / distance;
normaly = (ball2.getPosY() - ball1.getPosY()) / distance;

float tanx = -normaly;
float tany = normalx;

float dpTan1 = ball1.getSpeedX() * tanx + ball1.getSpeedY() * tany;
float dpTan2 = ball2.getSpeedX() * tanx + ball2.getSpeedY() * tany;

float dpNorm1 = ball1.getSpeedX() * normalx + ball1.getSpeedY() * normaly;
float dpNorm2 = ball2.getSpeedX() * normalx + ball2.getSpeedY() * normaly;

float m1 = (dpNorm1 * (ball1.getRadius() - ball2.getRadius()) + 2.0f * ball2.getRadius() * dpNorm2) / (ball1.getRadius() + ball2.getRadius());
float m2 = (dpNorm2 * (ball2.getRadius() - ball1.getRadius()) + 2.0f * ball1.getRadius() * dpNorm1) / (ball1.getRadius() + ball2.getRadius());

ball1.setSpeedX(tanx * dpTan1 + normalx * m1);
ball1.setSpeedY(tany * dpTan1 + normaly * m1);
ball2.setSpeedX(tanx * dpTan2 + normalx * m2);
ball2.setSpeedY(tany * dpTan2 + normaly * m2);
```

OBECNY ETAP PRAC

- Elementy działające poprawnie:
  - Okno programu
  - Okno symulacji
  - Jednostajnie opóźniony ruch kul
  - Zatrzymanie kul po odpowiednim wytraceniu prędkości
- Elementy nie działające:
  - Mechanizm wykrywania zderzeń (?)
  - Zmiana wektorów prędkości (kule się przesuwają wzajemnie, a nie odbijają)
  - Czas rzeczywisty, potrzebny do symulowania przyspieszenia kul (ujemnego)
- Elementy, które jeszcze nie powstały:
  - Biblioteki DLL
  - Wyliczanie czasu zderzeń

WNIOSKI WYCIĄGNIĘTE NA OBECNYM ETAPIE PRAC

- Wszystko da się zrobić, trzeba jedynie umieć znajdować błąd i go natychmiast naprawiać
- Symulacja prezentuje się najlepiej przy liczbie kul z zakresu <15,40>
  - Mniejsze wartości ograniczają liczbę zderzeń
  - Większe wartości powodują spore obciążenie komputera i zmniejszają płynność działania

DZIĘKUJĘ ZA UWAGĘ