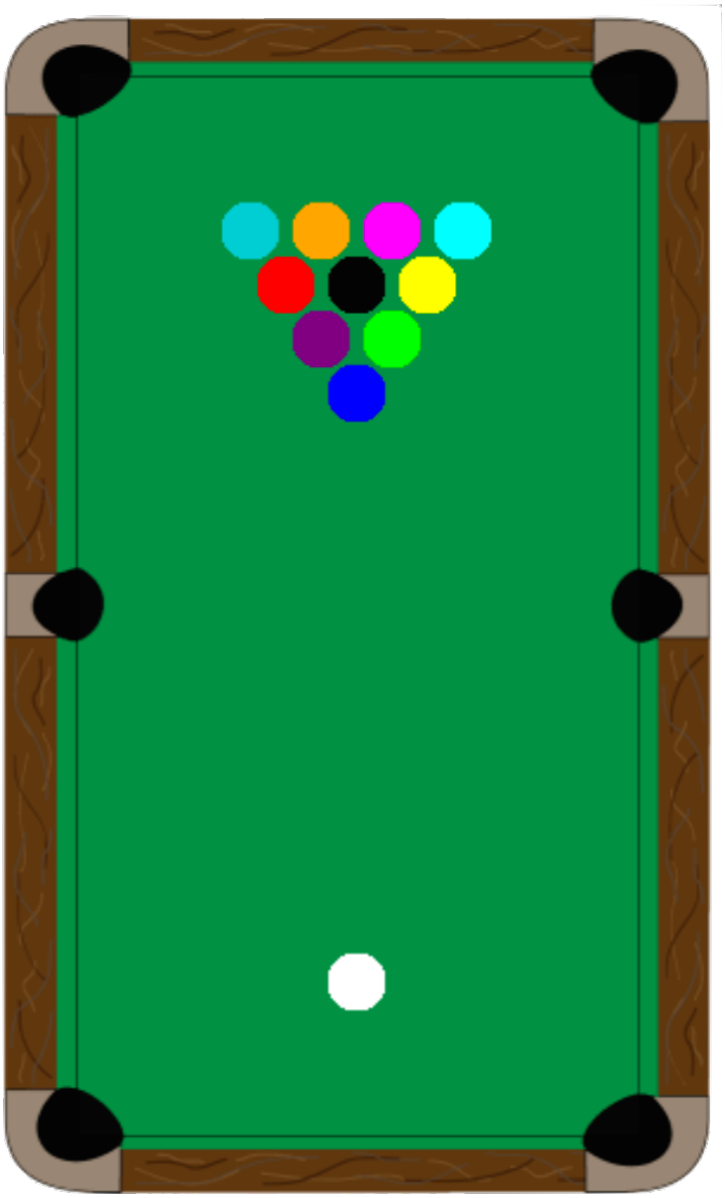
BILLARD

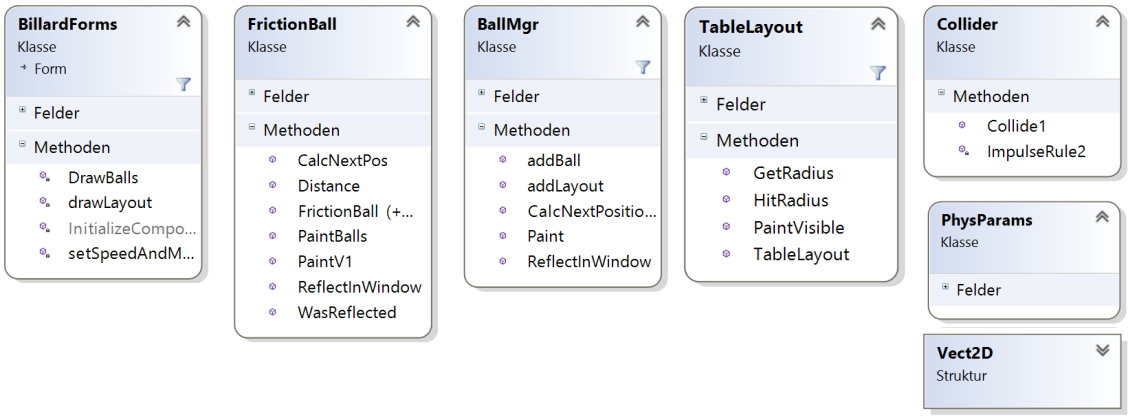


Lukas Ploder
5CHEL

Idee:

Mein Ziel war es, Billard in einer C# Windows-Forms Anwendung mithilfe diverser physikalischer Simulation unter Einfluss von Reibung, Stoßsatz und Masse zu realisieren.

Klassendiagramm



BillardForms

InitalizeComponents

Hier wird die Fenstergröße festgelegt, der Timer aktiviert, das Intervall festgelegt, sowie die Billardkugeln (Balls) als auch das Layout für die Löcher gezeichnet.

SetSpeedAndMass

Hier wird die Masse und die Geschwindigkeit der weißen Kugel festgelegt, als auch der Endpunkt des Queues (Position der Maus) ausgehend von der weißen Kugel gesetzt.

DrawBalls

Es werden über addBalls der Managerklasse die Billardkugeln gezeichnet.

DrawLayout

Es werden über addLayout der Managerklasse die Löcher gezeichnet gezeichnet.

FrictionBall

FrictionBall

Setzt die Position im Panel und die Farbe der Kugel.

PaintBalls

Zeichnen der Kugeln (Ellipsen).

PaintV1

Zeichnen des Zeigers der Kugeln, welcher die Geschwindigkeit, als auch die Richtung anzeigt.

CalcNextPos

Berechnung der neuen Position der Kugeln unter Einfluss von Reibung.

Distance

Berechnung des Vektors zwischen zwei Punkten.

WasReflected

Berechnung der Geschwindigkeit unter Einfluss des Stoßsatzes and der Bande.

ReflectInWindow

Überprüfung auf Bandenkontakt.

BallMgr

AddBall

Erstellt neues Object FrictionBall und fügt dieses in die _ballList ein.

AddLayout

Erstellt neues Object TableLayout und fügt dieses in die _LayoutListe ein.

CalcNextPosition

Berechnet die nächste Postion der Kugel über die CalcNextPos der FrictionBall-Klasse und überprüft auf Kollision mit weiteren Kugeln, bzw. auf Einlochen und löscht diese Kugeln gegebenenfalls.

ReflectInWindow

Überprüft über ReflectInWindow der FrictionBall-Klasse auf Reflexion an den Banden und berechnet gegebenenfalls die neue Geschwindigkeit in die neue Richtung.

Paint

Zeichnet sämtliche in der Liste befindende Kugeln, sowie das Layout über die Paintklasse der FrictionBall-Klasse bzw. die der TableLayout-Klasse.

TableLayout

TableLayout

Setzten der Koordinaten, der Größe und der Farbe.

PaintVisible

Zeichnen der Löcher.

GetRadius

Gibt Radius der Löcher für die Berechnung der HitRadius-Methode zurück.

HitRadius

Überprüft ob eine der Kugeln in die Löcher fallen.

Collider

Collide1

Führt Überprüfung und Berechnungen für die Kollision zwischen zwei Kugeln aus. (Stoßsatz).

ImpulseRule2

Vertauscht die Vx-Komponenten, die Vy-Komponenten bleiben unverändert.

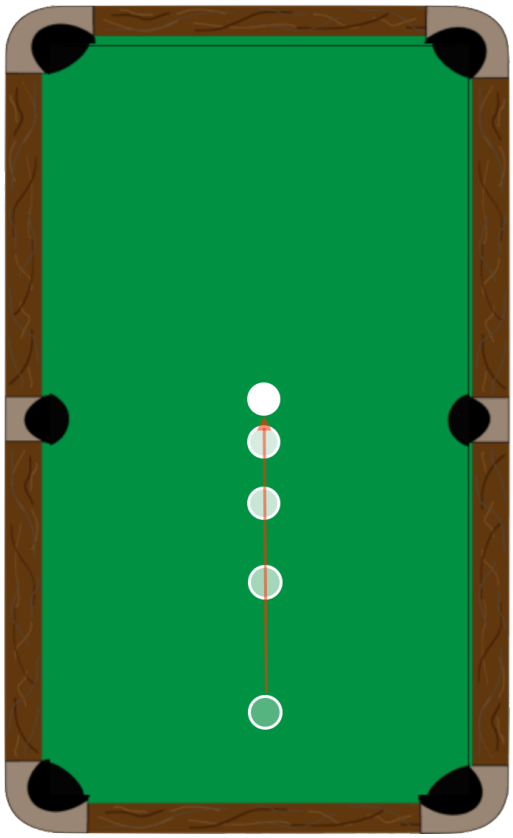
PhysParams

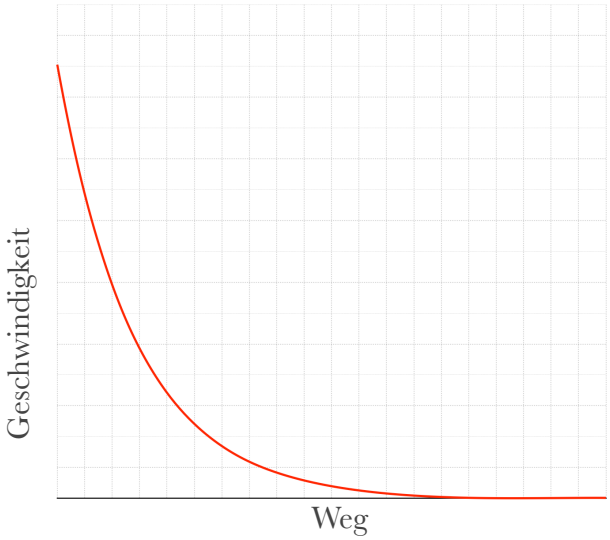
Beinhaltet sämtliche Konstanten welche für die Berechnung der Simulationen notwendig sind (Reibungskonstanten, Reibungskonstante bei Reflexion, Timerintervall etc...).

Vect2D

Hier befinden sich alle Methoden für die Vektorberechnung (Stoßsatz, Reibung etc...).

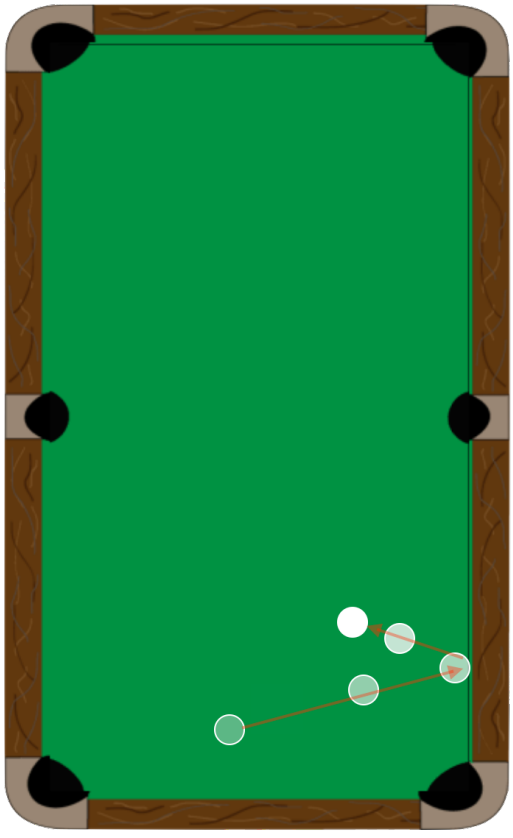
Reibung



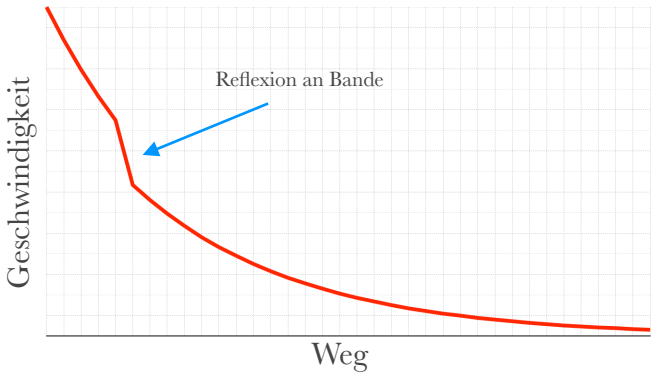


Die Geschwindigkeit der Kugel nimmt durch die Reibung exponentiell nach folgender Gleichung ab:

$$v(v_{n-1}) = v_{n-1} * (1 - 0.01 * \frac{1}{IterPerTick})$$



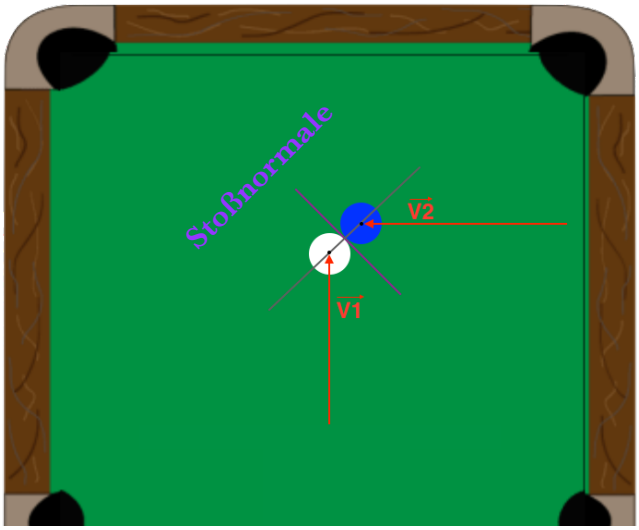
Bei der Reflexion an einer Bande nimmt die Geschwindigkeit zusätzlich zur Reibung noch um eine Weitere Konstante ab:



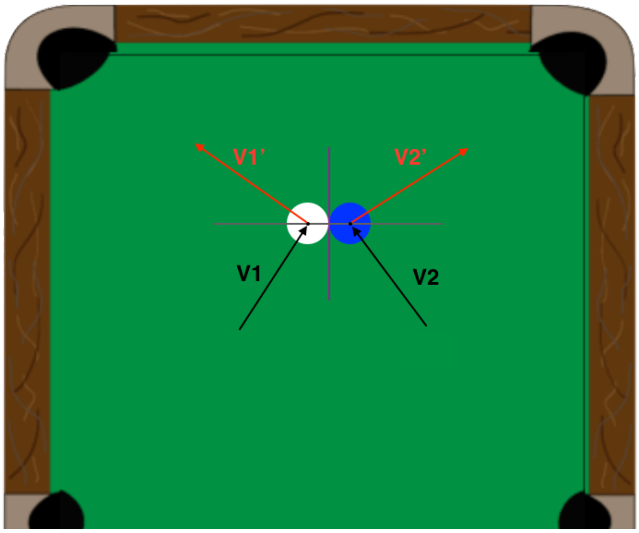
Stoßsatz

Treffen zwei Kugeln aufeinander so wird der indirekte Stoß ausgeführt. Dies funktioniert folgendermaßen:

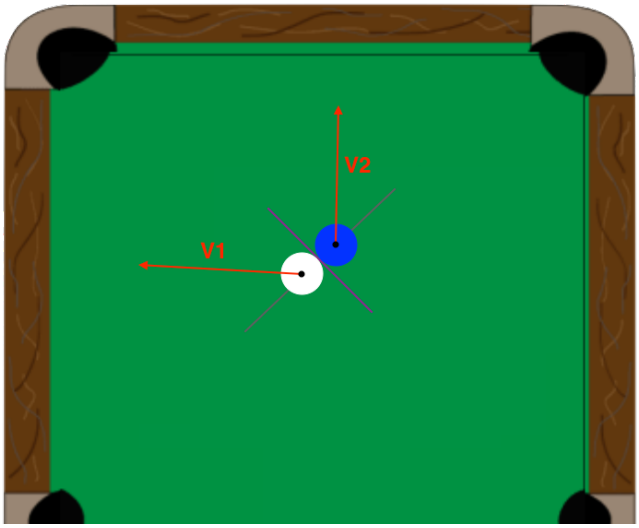
Zuerst wird die Stoßnormale so verdreht, dass sie zur Y-Achse wird.



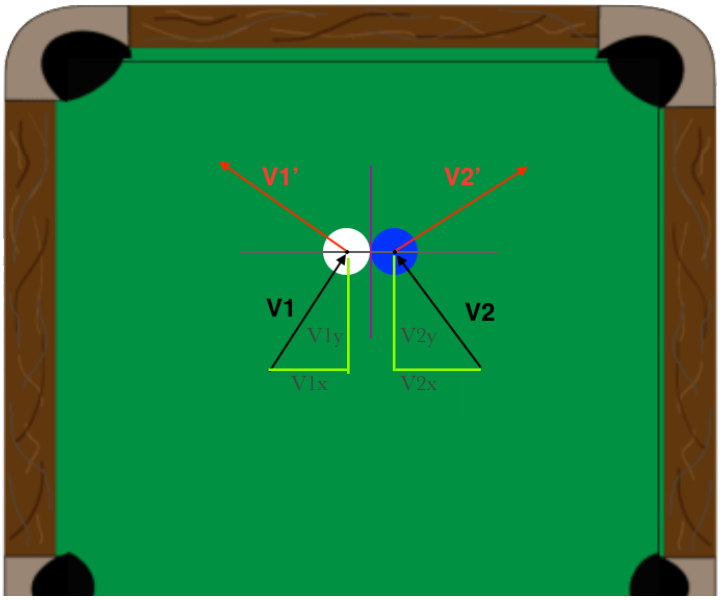
X Komponenten werden ausgetauscht, Y-Komponenten bleiben gleich.



Danach wird die Stoßnormale wieder in ihre Ausgangsposition zurückgedreht.



Somit ergibt sich folgendes:



V1x' = V2x

V1y' = V1y

V2x' = V1x

V2y' = V2y

Stoßsatz in C#:

```
public static void Collide(FrictionBall aB1, FrictionBall aB2)
{
  // Vektor von B1 nach B2
  Vect2D r1 = Vect2D.VectBetweenPoints(aB2.pos, aB1.pos);

  // vektor zum Zurückdrehen
  r1 = r1.GetNormalizedVersion();

  // vektor zum Vorwärtsdrehen
  Vect2D r2 = r1.GetComplexConjugate();

  // V-Vektoren so drehen, daß die Stoßnormale mit der Y-Achse zusammenfällt
  aB1.V.CoMultTo(r2); aB2.V.CoMultTo(r2);

  // Vx-Komponenten austauschen Vy-Komponenten bleiben unverändert
  ImpulseRule(aB1, aB2);

  // V-Vektoren wieder zurückdrehen
  aB1.V.CoMultTo(r1); aB2.V.CoMultTo(r1);
}

//  Austauschen der Vx Komponenten
static void ImpulseRule(FrictionBall b1, FrictionBall b2)
{
  int nenner = b1.m + b2.m;
  double v1s, v2s;
  v1s = ((b1.m - b2.m) * b1.V.X + 2 * b2.m * b2.V.X) / (double)nenner;
  v2s = (b1.m * b1.V.X) / (double)nenner;
  b1.V.X = v1s;
  b2.V.X = v2s;
}
```