ikdoeict.be

# Game Programming

## Workshop 2: Fysische simulatie

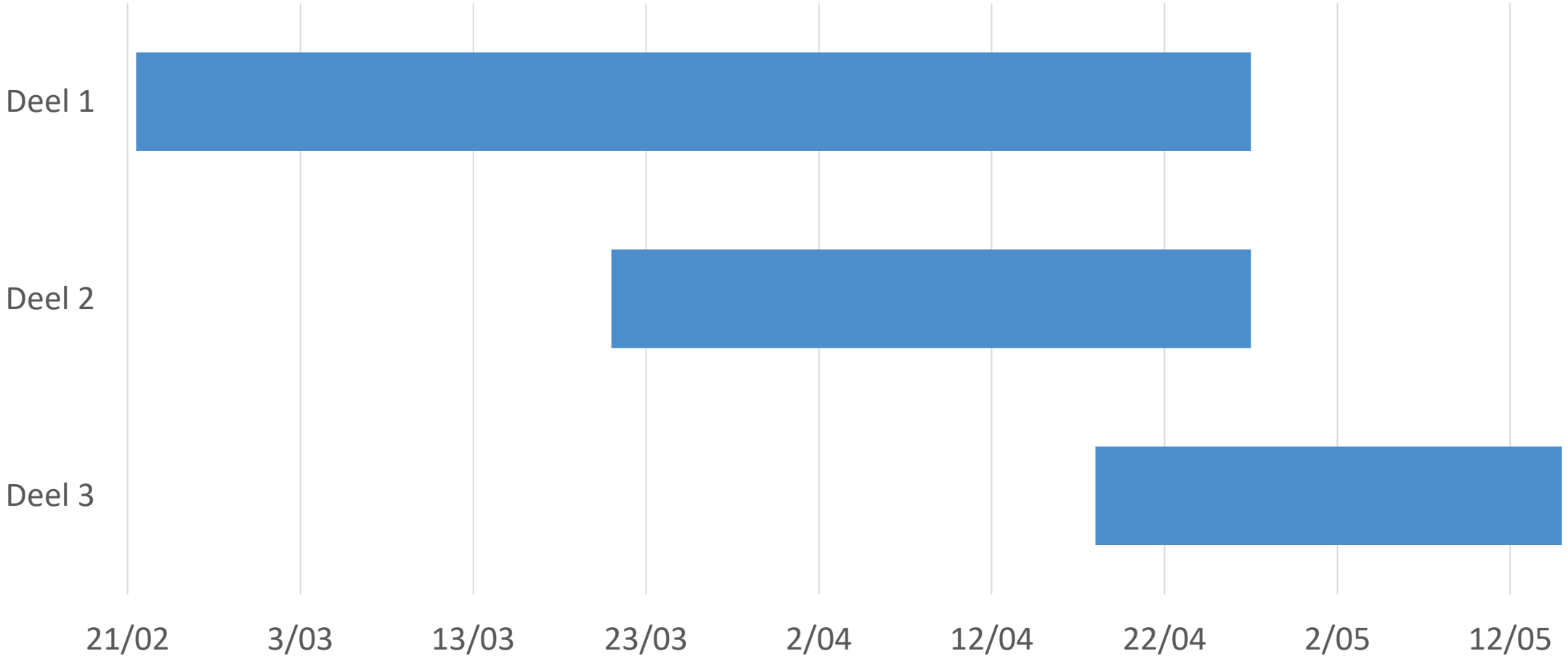
LID VAN ASSOCIATIE KU LEUVEN

# Overzicht: GP workshops en deadlines

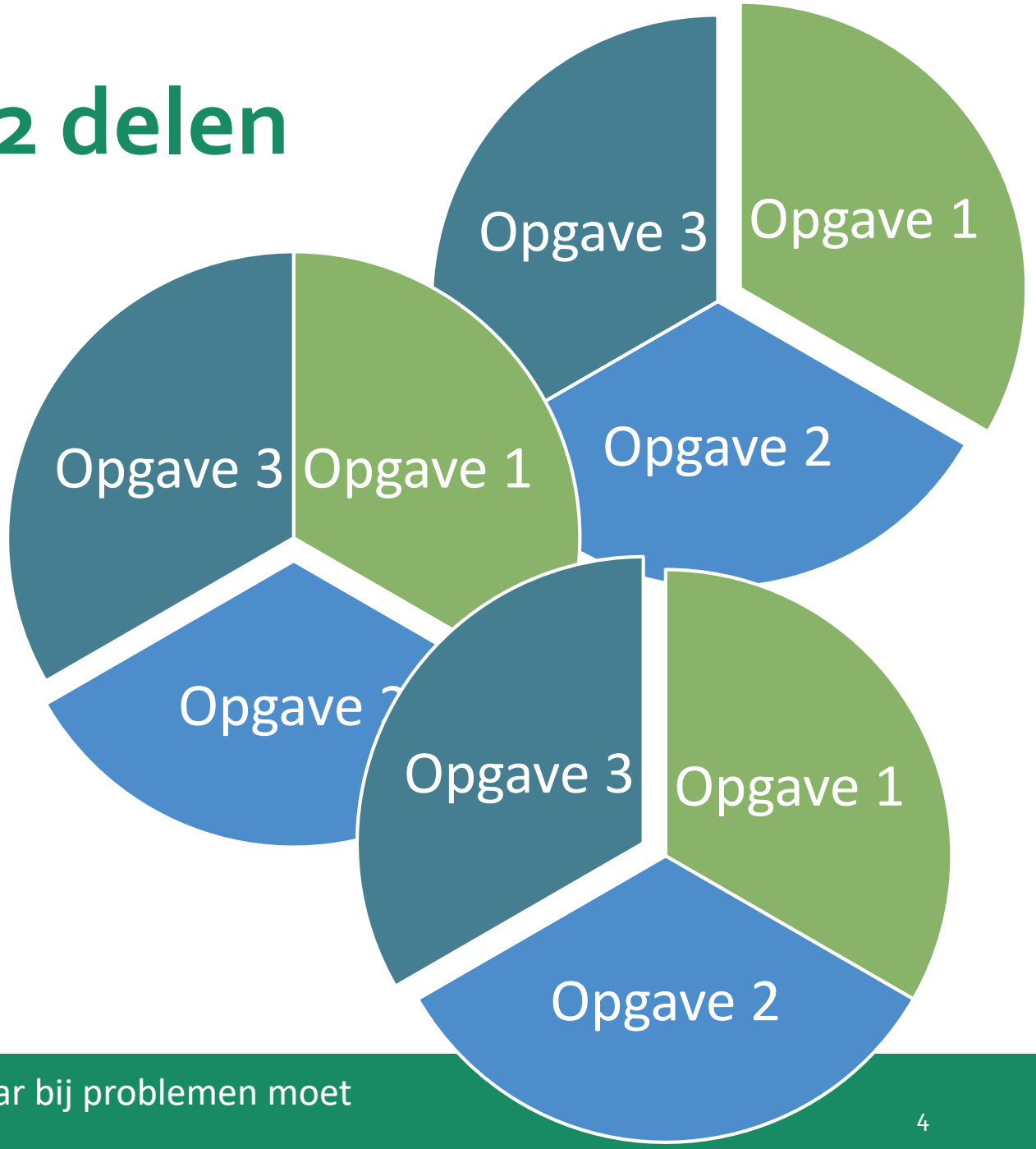
- 1 onderwerp - 3 delen - 3 workshops
- Vandaag workshop 2
- Wo 18/04
  - 11u20 workshop 3
- Vr 27/04
  - 20u00 deadline deel 1 & 2 -> clone 1 van je GIT project
- Vanaf week van 1 mei: verplichte feedbacksessies
  - inschrijven via Tolinto!
  - verdediging van project 1 & 2 (clone 1)
- Vrij 25/5
  - 20u00: deadline (update deel 2 + *) deel 3 -> clone 2 van je GIT project
- juni-examen : verdediging deel 3 + eventueel deel 2 * (clone 2)
  - inschrijven via Tolinto in de juiste groep! (zie examenrooster)

* wordt ten stelligste afgeraden

# Overzicht



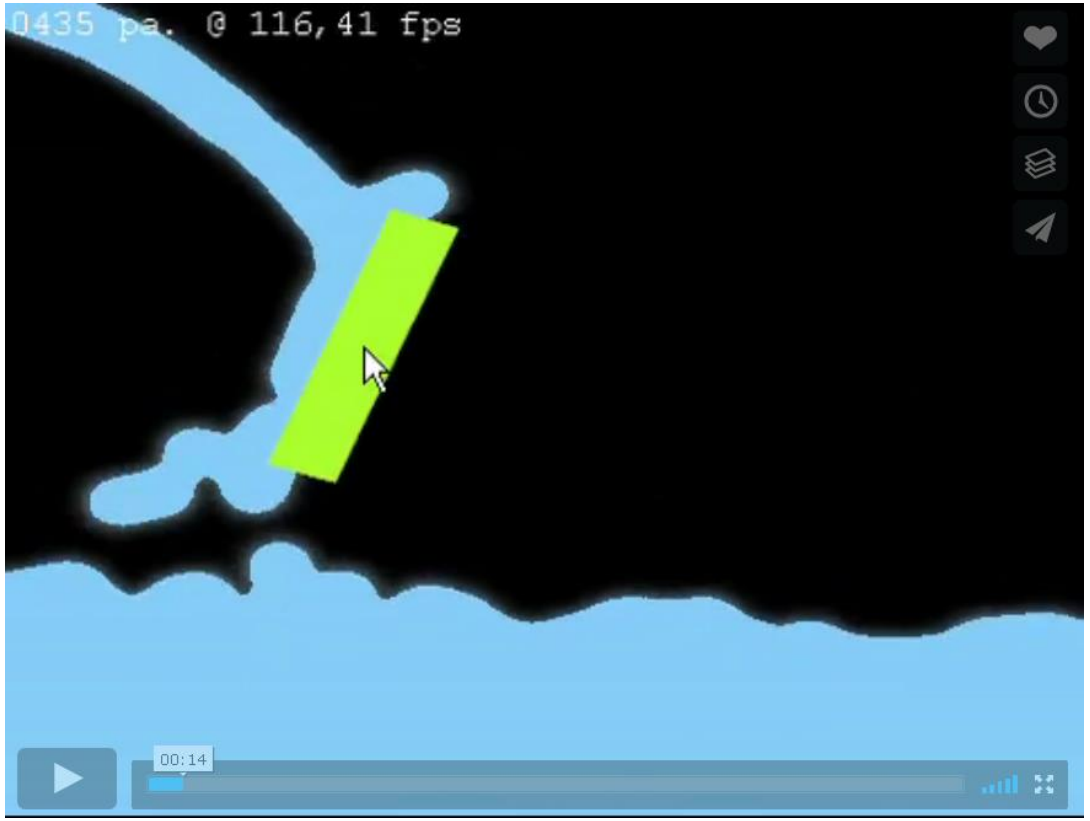
# Combineer minstens 2 delen


Uiteraard moeten de 3 delen wel gemaakt worden, maar bij problemen moet één systeem niet verplicht geintegreerd worden

# Een kleine demo

- 2DFLUID demo C# OpenGL/OpenTK project
- http://vimeo.com/4391370



# Inhoud

- Deel 1: 2D computer graphics
  - Raamwerken voor computer-Graphics
  - Coördinatenstelsels & -transformaties
- Deel 2: Basic game Physics
  - Basiswetten uit Fysica: beweging
  - Werken in 2 dimensies, ontbinden van vectoren
  - Krachten – impuls - wetten van Newton – gravitatie – arbeid/energie/vermogen – wrijving - botsingen
- Deel 3: Physics-based' simulatie
  - Algemene structuur computergame/simulatie
  - Case study: moving ball
  - analytische methode vs. Euler
- Deel 4: Opgave (eerste deel)

Deel 1

# 2D COMPUTER GRAPHICS

# Enkele frameworks voor grafische toepassingen

- GDI+ (Windows): 2D, niet erg performant, eenvoudig.
- WPF (Windows): 2/3D, performanter en krachtiger

- OpenGL (Silicon Graphics, Open Graphics Library…):
  - Enkel graphics
- DirectX (Microsoft):
  - Vrij complex, zeer krachtig & performant, low level
- XNA: 'High Level framework voor gaming'
  - Krachtig, snel resultaat, eigenzinnig
- Flash/Silverlight
  - Snel resultaat, beperkte mogelijkheden en performantie

- Eigen framework
  - Vb: Unreal Development Kit, CryEngine
  - Gebruik maken van OpenGL, DirectX, …

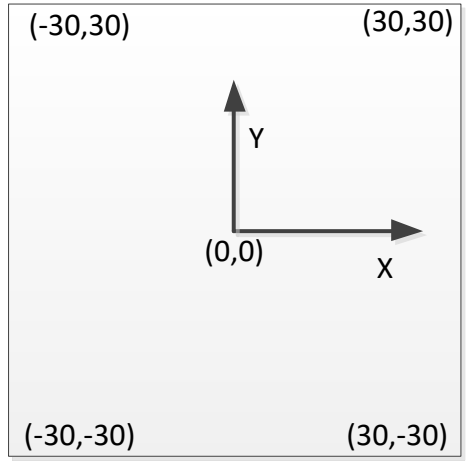
# World Coördinate System (WCS)

Natuurlijk, logisch coördinatenstelsel voor de toepassing.

Voorbeeld: grafische weergave van de functie

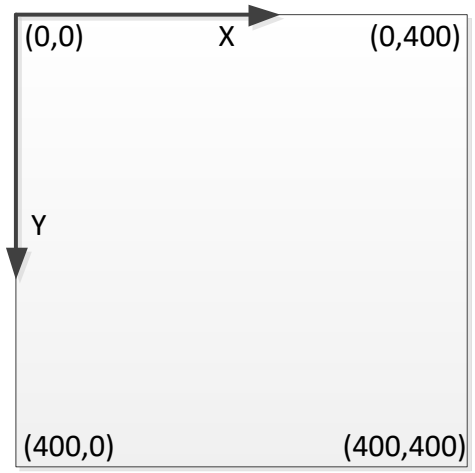
$$y(x)=x^2+x-6 \text{ voor } x \in [-5, 5]$$

==> y is maximaal 24, mogelijke keuze:



# Device Coördinate System (DCS)

- coördinatenstelsel zoals vastgelegd door het grafisch 'device':
- Voorbeeld: picturebox van 400x400 pixels:



# Coördinatentransformaties

Onze code genereert logischerwijze coördinaten in WCS

Coördinaten in DCS (voor rendering).

Voorbeeld:

oorsprong (0,0) in WCS

(200,200)

Omvorming moet gebeuren voor elke punt met coördinaten in WCS.

# Coördinatentransformaties

Steeds minimaal twee delen:

- Schaling:

  X * 400 / 60  Y * 400/60 * (-1)

  (vanwege tegengestelde zin)

- Verplaatsing

  X + 30  Y - 30

- (soms rotatie)

# Coördinatentransformaties

- Reeds voorzien in Graphics framework:

```
display = pictureBox1.CreateGraphics();
          // transformatie voor display
          float SchaalX = pictureBox1.Width / 60.0f;
          float SchaalY = pictureBox1.Height / 60.0f;
          display.ResetTransform();
          display.ScaleTransform(SchaalX, -SchaalY); // schaling
          display.TranslateTransform(30f, -30f);      // oorsprong


         …

 Pen p = new Pen(Color.Blue, 0.05f);
 display.DrawLine(p, new Point(-30, 0), new Point(30, 0));
 display.DrawLine(p, new Point(0, -30), new Point(0, 30));
```

Deel 2

# BASIC GAME PHYSICS

# Newtoniaanse Mechanica

Rechtlijnige beweging (1 dimensie)


- Snelheid en positie

$$v = \frac{x_1 - x_0}{t_1 - t_0} = \frac{\Delta s}{\Delta t} \qquad v(t) = \lim_{\Delta t \to 0} \frac{\Delta s}{\Delta t} = \frac{ds}{dt} \qquad x(t) = x_0 + \int_{t_0}^{t} v(t)dt$$

- Versnelling

$$a_{gem} = \frac{v_1 - v_0}{t_1 - t_0} = \frac{\Delta v}{\Delta t} \qquad v(t) = v_0 + \int_{t_0}^{t} a(t)dt \qquad a(t) = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{dv}{dt} = \frac{d^2 s(t)}{dt^2}$$

# Newtoniaanse Mechanica

Speciale gevallen

- Eenparig Rechtlijnige beweging (1 dimensie)

$$v(t) = Cte \qquad s(t) = x(t) - x_0 = \int_{t_0}^{t} v(t)dt$$

$$v\,(t - t_0) = v\,t \;\; (als \;\; t_0 = 0)$$

- Eenparig versnelde beweging (1 dimensie)

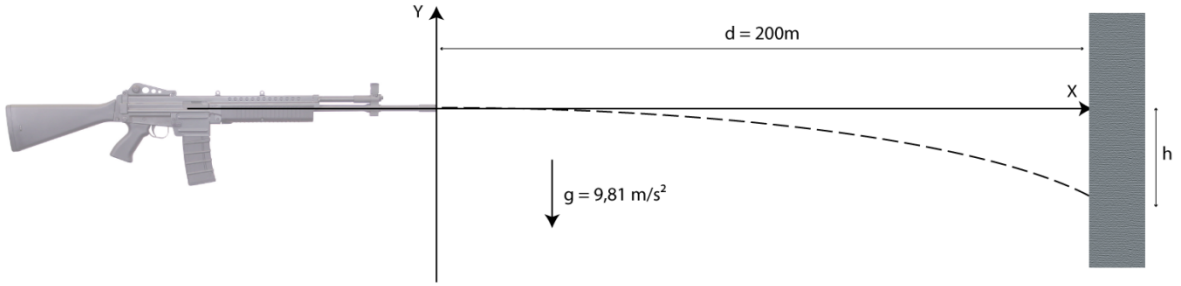
$$a(t) = Cte \qquad s(t) = x(t) - x_0 = \int_{t_0}^{t} v(t)dt$$

$$= v_0.(t - t_0) + \frac{a.(t - t_0)^2}{2}$$

als $t_0$ = o

$$v(t) = v_0 + a.t \qquad x(t) = x_0 + v_0.t + \frac{a.t^2}{2}$$

# Lineaire beweging (2 dimensies)




- *Ontbinden in 2 componenten (X, Y)*
- $a_x = 0, \qquad v_x = \frac{700m}{s}, \qquad t = \frac{200m}{v_x} = 0{,}286s$
- $a_y = -9{,}81m/s^2$ na *t:* $d_y = \frac{(a_y.t^2)}{2} = 0{,}4m$

# Massa, snelheid en kracht
# De tweede wet van Newton

- Krachten zijn de oorzaak van beweging, vervorming, breking, …
- Eenheid van kracht : Newton : 1N=1kg m/s²
- 1N geeft aan een massa van 1 kg een versnelling van 1m/s².
- Verschillende types: gravitatie, elektrische/magnetische aantrekking, wrijving, door druk die inwerkt op een lichaam (bv. in verbrandingsmotoren).
- massa van een voorwerp is een maat voor de kracht die nodig is om het voorwerp een bepaalde versnelling te geven:

$$F = \frac{d}{dt}(m.v) \quad of \quad F = m\,.a$$

Als m onafhankelijk is van de tijd

# De Wetten van Newton

- **Wet 1** : Een vrij deeltje beweegt met constante snelheid zonder versnelling (traagheidswet, inertiewet)

$$\mathbf{F} = 0 \text{ implies } \mathbf{a} = 0$$

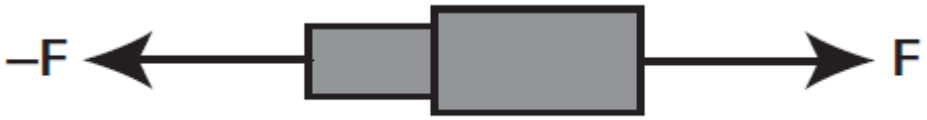
$$\mathbf{v} = \text{constant}$$

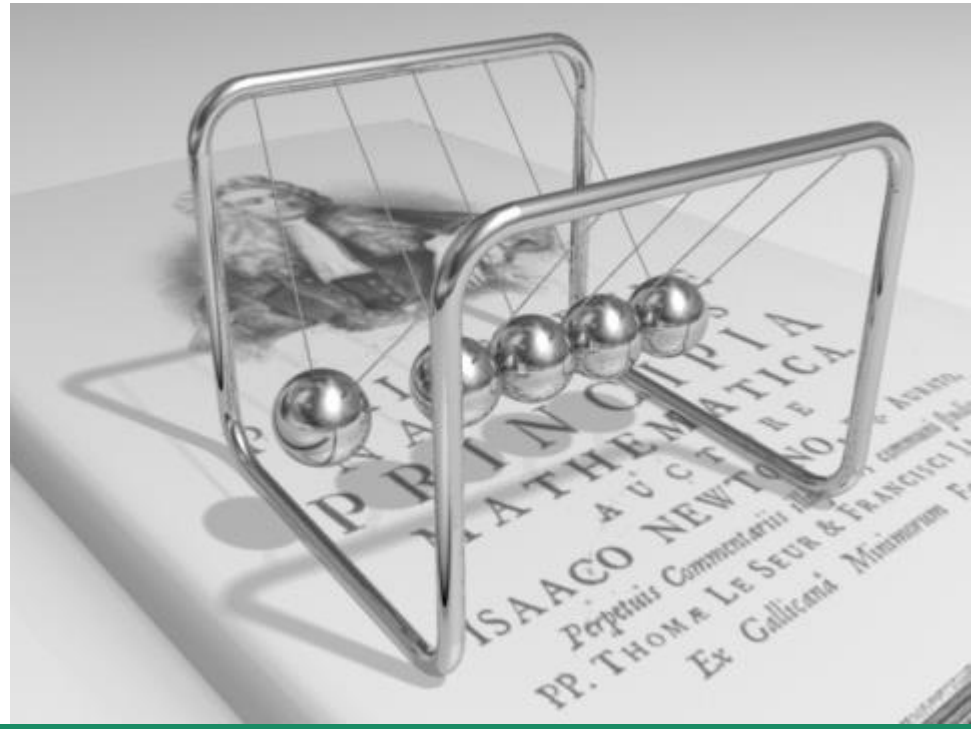
# De Wetten van Newton

- **Wet 2** : De verandering van de beweging is evenredig met de kracht en volgt de rechte lijn waarin de kracht werkt

# De Wetten van Newton

- **WET 3** : Als een lichaam een kracht op een ander lichaam uitoefent, oefent dit laatste lichaam een even grote maar tegengestelde kracht uit op het eerste



# Collisions // Botsingen

Tussen botsende voorwerpen wordt impuls en energie uitgewisseld.

- Impuls is een alternatieve manier om de toestand van een object in beweging uit te drukken. De grootheid *m . v* wordt de impuls *p* (of hoeveelheid van beweging) genoemd

$$\vec{p} = m\,\vec{v}$$

- En volgens 2de wet van Newton, bij cte massa $\vec{F} = \frac{d}{dt}(m.\vec{v}) = \frac{d}{dt}\vec{p}$

- **Wet van behoud van impuls**: in een geïsoleerd stelsel is de som van de impuls van alle deeltjes steeds constant

$$P = \sum_i p_i = p_1 + p_2 + p_3 + \cdots = Cte$$

# Elastische Botsingen

- Bij elastische botsingen zijn zowel de totale impuls als de totale kinetische energie constanten van de beweging

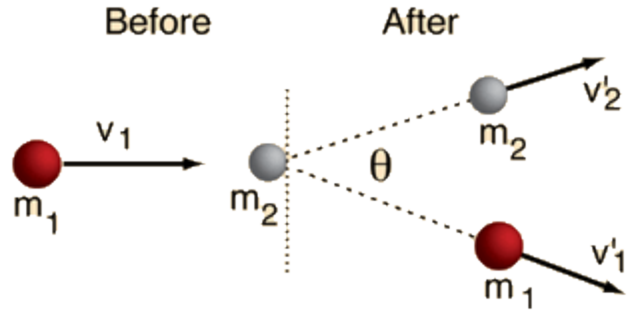
$$(m_1.\vec{v}_1 + m_2.\overrightarrow{v_2}) \; voor \; botsing = (m_1.\vec{v}_1 + m_2.\overrightarrow{v_2}) \; na \; botsing$$

en

$$E_{kin} : \left( \frac{m_1.{v_1}^2}{2} + \frac{m_2.{v_2}^2}{2} \right) voor \; botsing = \left( \frac{m_1.{v_1}^2}{2} + \frac{m_2.{v_2}^2}{2} \right) na \; botsing$$

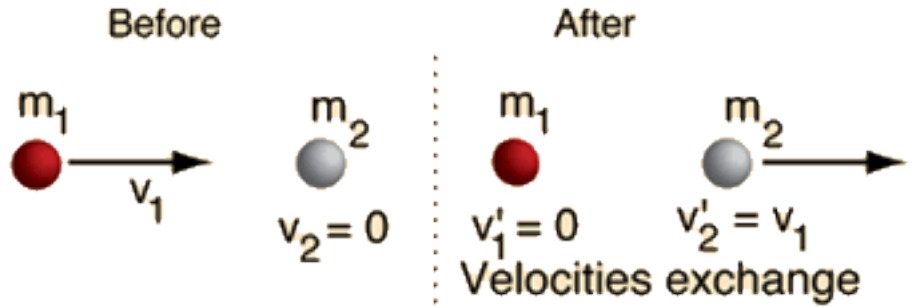
- Het verschil in snelheid voor en na de botsing is  gelijk, maar de richting van de relatieve snelheid is omgekeerd

# Elastische Botsingen



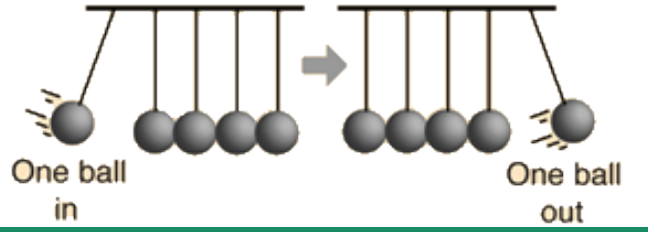

Case I: $m_1 = m_2$

If not head-on then $\theta = 90°$



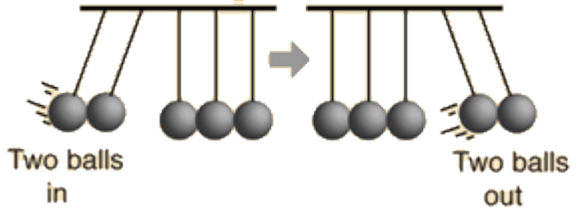

Momemtum in: mv = momentum out

Kinetic energy in: $\frac{1}{2}mv^2$ = kinetic energy out

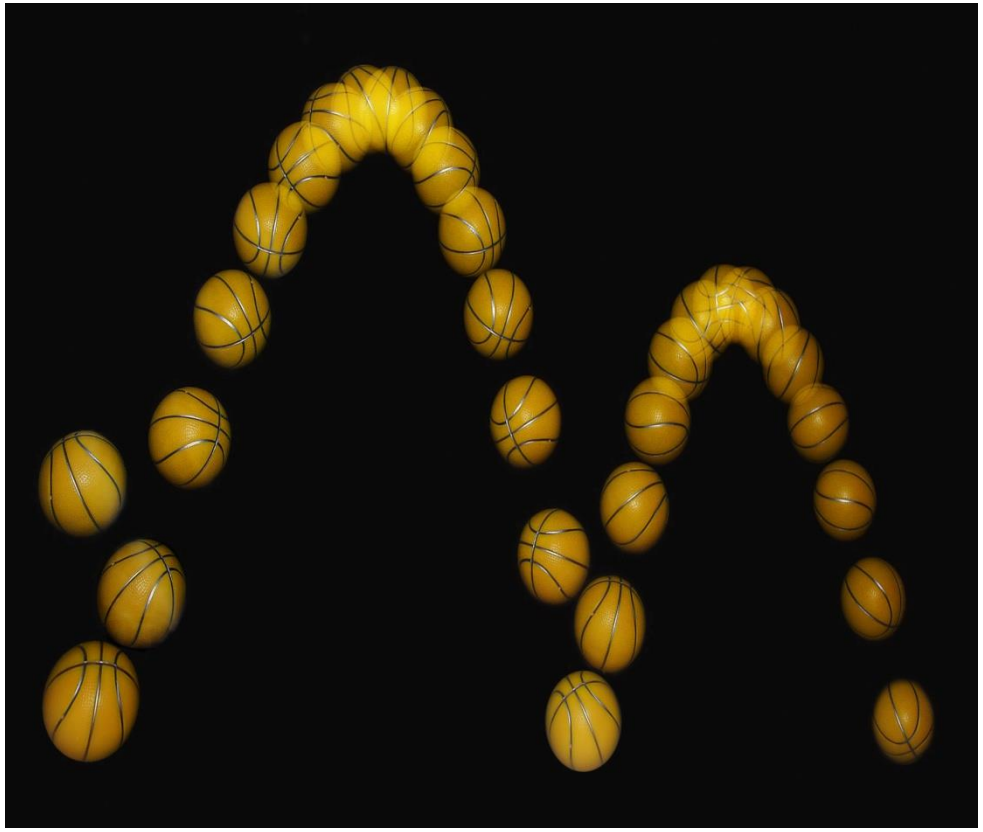



Momemtum in: 2mv = momentum out

Kinetic energy in: $\frac{1}{2}2mv^2$ = kinetic energy out



# Niet perfect elastische botsingen



**De bal zal na elke botsing een lagere hoogte bereiken.**

**Bij een niet elastische botsing zal de relatieve snelheid verminderen met factor e**

**met voorwaarde: 0 < e < 1**

$$v_{1na} - v_{2na} = -e\left(v_{1voor} - v_{2voor}\right)$$

# Massa vs Gewicht

- de gravitatiekracht: de aantrekkingskracht die voorkomt tussen alle massa's

$$F = G.\frac{m_1.m_2}{r^2}$$

met G = 6.67390e−11 N.m²/kg².

$$F = G.\frac{m_{aarde}}{r^2}.m = 6{,}67390.10^{-11}.\frac{5{,}9736.10^{24}}{(6{,}375.10^{6})^2}.m = 9{,}81.m\ N$$

Valversnelling
g=9,81 m/s²

- Massa = hoeveelheid materie
- Gewicht = de kracht die een lichaam uitoefent op een steunpunt (of ophanging) o.a. ten gevolge van de gravitatie
- Vb in de ruimte: Feather and hammer Drop on the moon: http://www.youtube.com/watch?v=5C5_dOEyAfk

# Vraagstukken

- Moeder en dochtertje dragen samen een boodschappentas, elk aan één lus van 40cm, maar door haar korte armpjes slaagt de kleine er alleen in horizontaal aan haar lus te trekken. De tas weegt 180N, en het dochtertje trekt 45N; hoeveel draagt de moeder?
- Iemand hoort de donder juist 5 sec nadat hij de bliksem heeft gezien. De geluidsnelheid is 340 m/s en de lichtsnelheid 300.000 km/s.  Hoe ver staat hij van de blikseminslag?
- Drie maten zijn op boottocht. Plots horen zij het geruis van een waterval naderen en springen in paniek uit de boot om zich te redden. Ze kunnen niet alle drie evengoed zwemmen: de eerste haalt 6km/h, de tweede 5km/h en de derde 3,5km/h. Het water is 30 m breed en stroomt aan 3m/s. De waterval is nog 35m ver en de boot vaart in het midden. Wie haalt het en wie niet?

# Vervolg Newtoniaanse Mechanica

Zie verder in cursusnota's:

- Krachten
- Impuls
- Wetten van Newton
- Gravitatie
- Arbeid/energie/vermogen
- Wrijving
- Botsingen

*Geen formules of bewijzen kennen, wel inzicht en kunnen gebruiken*

(vermeld wel in je verslag wat je nodig hebt)

Deel 3

# PHYSICS-BASED' SIMULATIE

# Algemene structuur computergame

- User interactie (sturen, acties, aanpassen parameters, manipulatie van objecten)

- Render-loop
  - Visualisatie van de actuele wereld
  - Genereert telkens één frame
  - Moet voldoende dikwijls uitgevoerd worden voor een aanvaardbare framerate (> 25Hz)

- 'Game Loop'
  - Collision detection: detecteren van interactie tussen objecten
  - Berekenen van de nieuwe toestand voor de objecten in de 'wereld' (op basis van bv. de wetten uit de fysica, AI)

- Timing: (voldoende nauwkeurig) timing-mechanisme is meestal vereist

# Case Study: JoPhysics.sln

# Analytische methode
# Kinematica

Op voorhand alle (differentiaal) vergelijkingen oplossen en de positie van elk object vastleggen als functie van de tijd

- Bv.: eenparig versnelde lineaire beweging:

$$s(t) = s_0 + v_0.t + \frac{a.t^2}{2}$$

- In de game-loop op basis van de tijd de huidige positie berekenen voor elk object

# methode van EULER

De differentiaalvergelijkingen worden niet op voorhand opgelost, maar worden  rechtstreeks gebruikt in de code om telkens de veranderingen in de versnelling, snelheid, positie enz… opnieuw te berekenen aan de hand van het tijdsinterval tussen twee berekeningen.

- Bvb Assumptie: tussen twee berekeningen is de snelheid constant

```
public void Integrate(particle p, float deltaT)
{
    p.positie += p.snelheid * deltaT;
    p.snelheid += p.versnelling * deltaT;
    p.versnelling = (1f / p.massa) * p.kracht;
}
```

# methode van EULER

veronderstelling kan te grote fout geven:

- veronderstel dat de versnelling a(t) tussen 2 frames constant blijft:

- In de 'game-loop' op basis van de tijd sinds vorig frame nieuwe positie, snelheid en eventueel versnelling berekenen.

```
public void Integrate(particle p, float deltaT)
{
    p.positie += p.snelheid * deltaT + (deltaT * deltaT / 2f) * p.versnelling;
    p.snelheid += p.versnelling * deltaT;
    p.versnelling = (1f / p.massa) * p.kracht;
}
```

# methode van EULER

**Probleem:**
**Er wordt nog steeds een fout gemaakt (vermits a(t) niet noodzakelijk constant is)**

==> enkel geschikt voor kleine versnellingen/wijzigingen t.o.v Δt

- Voorbeeld: jet (v =1000Km/h)
  Bij een framerate van 100Hz is Δt=10ms en legt de jet tussen 2 frames
  2,8m af, of na 1 sec 280m

- Met een versnelling van 5m/s² wordt dit 285m. Als we de snelheid constant houden tussen 2 frames maken we dus een fout van 5m op 1 seconde!

Oplossing: 'methode van Runge Kutta' (stuk complexer)

Deel 4

# OPDRACHT GAME PHYSICS

# 2D fysische simulatie

Benadering:

- Eigenlijk 3D bord
- Maar kan benaderd worden door 2D fysica
  - Geen opstuiteren van de bal door draaien van bord
  - Zwaartekracht ontbinden in 2 componenten:
    - Deel loodrecht op bord: genereert wrijving of kan genegeert worden (keuze)
    - Deel evenwijdig met bord: geneert versnelling voor de bal
  - Eventueel extra coordinatensysteem gebruiken

# 2D fysische simulatie

- Fysische eigenschappen van de muren en de bal:
  - Versnellen (zwaartekracht), snelheid, positie
  - Botsingen van de bal tegen de muur
  - (wij willen fysica formules in je code zien staan)
- Een game-loop die de fysica voldoende frequent uitvoert
- Starten/stoppen/herstarten van het spel

# TIP

Stel je model incrementeel op en implementeer ook incrementeel. Stuur werkende versies telkens door naar GIT. Zo heb je steeds een werkende versie die je kan verfijnen en realistischer maken.

# Vereisten

**Code**

- Alle code in C#
- Alles moet werken op een labo PC
- Geen extra bibliotheken toegestaan
- Ook niet voor de fysica

**Documentatie**

# Vereisten

**Code**

**Documentatie**

- Bespreek de verschillende stappen en elementen in je programma
- Bespreek de nodige fysica theoretisch en in je programma
- Verklaar je keuzes
- Bespreek je experimenten

# Structuur

**Code**

- 1 Solution voor de volledige OLA met verschillende projecten

**Documentatie**

- 1 Verslag voor de volledige OLA
  - Verschillende onderdelen in het verslag
    - Voorstudie opsplitsen
    - Praktische uitwerking opsplitsen