# Obligatorisk uppgift: Studsande objekt

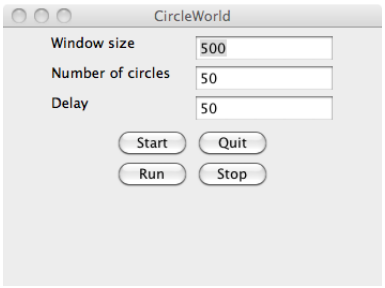
**Moment:** Grafik, grafiska användargränssnitt, händelsestyrda program.

**Förberedelser:** Gå igenom de tre nätlektionerna om grafik, grafiska användargränssnitt och rörliga figurer.

## Övergripande

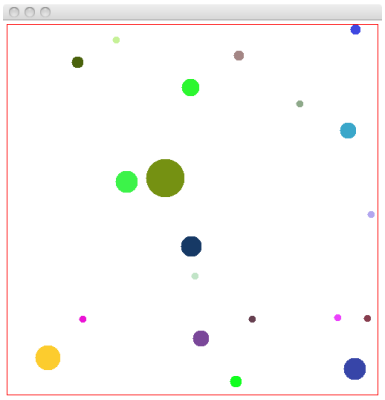
Ni skall skriva ett program som simulerar objekt (t.ex. bollar) som rör sig i en låda. Det skall finnas ett grafiskt användargränssnitt för att specificera vissa egenskaper (t.ex. antalet bollar, bollarnas radie, ...) samt för att starta, stoppa, mm.

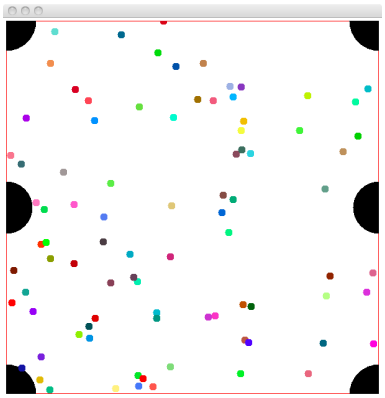
Exempel på det grafiska användargränssnittet :




Ni får själva hitta på mer exakt vad som skall hända i lådan. Några förslag finns längre ner.

Förloppet skall illustreras grafiskt i ett separat fönster. Exempel på två möjliga ögonblicksbilder:





## De rörliga objekten

Vi kallar objekten för bollar men ni kan tänka er andra typer (atomer, sköldpaddor, rävar och kaniner, människor, ...). Bollarna skall ha en *position* (en vektor i x-y-planet), en

*hastighet* (också en vektor i x-y-planet), en *storlek* (t.ex. en radie) och en *färg*. Ni får lägga till flera egenskaper.

Objekten skall röra sig i hastighetsvektorns riktning. Denna kan vara konstant så länge objekt inte kolliderar med något. Det är dock tillåtet att förändra hastighetsvektorn riktning så att en mer ”random walk” åstadkoms eller att objekten utsätts för gravitation och/eller friktion.

Vid kollision med vägg skall objekten studsa enligt vanliga regler (”utfallsvinkeln lika med infallsvinkeln”).

Exakt vad som händer när två objekt kolliderar får ni bestämma själva men *något* skall hända! Exempel:

- De kan studsa mot varandra (typ biljardbollar). Formel finns nedan.
- De kan slås samman till en större boll (typ såpbubblor)
- Det ena objektet kan förinta den andra (typ rävar och kaniner)
- De två objekten kan, under rätta omständigheter, producera ytterligare ett objekt (typ sköldpaddor)

Ett villkor: Antalet objekt skall förändras under körningen. I biljardfallet betyder det att man måste ha minst ett hål där bollarna försvinner.

Objekten kan också förses med en *ålder* som gör att de kan dö — t.ex. med ökande sannolikhet med ökande ålder.

## Minimikrav på det grafiska användargränssnittet

Det skall åtminstone finnas knappar för att starta och avbryta. Man kan också ha knappar för att pausa och återstarta (frivilligt)

Det skall finnas textfält åtminstone för att specificera

- antalet objekt
- antal millisekunder mellan flyttning/omritning

Fälten ska vara försedda med defaultvärden.

Fälten ska avläsas när man trycker på startknappen. Man ska alltså kunna starta en ny simulering med nya värden utan att starta om programmet.

## Implementation

### En klass `Vector`

Eftersom position och rörelser liksom krockar både med väggar och mellan objekt uttrycks bäst med hjälp av vektorer så *skall* ni skriva en klass `Vector`. Det är enklast att uttrycka vektorerna i reella koordinater på intervallet [0,1] och sedan låta ritmetoderna avbilda dessa på fönsterkoordinaterna. (Det innebär t.ex. att ett objekt *vet* att den är nära högerväggen när positionsvektorns x-koordinaten är nära 1)

Din vektorklass ska uppfylla denna specifikation.

Observera att `Vector`-objekten ska vara ”oföränderliga” dvs ingen metod ska ändra på

instansvariablerna!

Ett tips: Metoden `angle` har nytta av metoden `Math.atan2`!

Skriv en `main`-metod i klassen som testar metoderna!

### En klass för objekten

Klassen `Ball` (eller `Circle` eller `Turtle` eller ...) samlar de egenskaper som individuella bollar/cirklar/paddor har.

Minimalt ska klassen ska ha två instansvariabler av typen `Vector` för att representera position och hastighet. Andra troliga egenskaper är färg och storlek.

Klassen behöver bland annat `get-` och `set`-metoder för attributen samt en metod `move` som förflyttar den i hastighetsvektorns riktning. Metoden hanterar lämpligen också krockar med väggar samt eventuella slumpmässiga förändringar av hastighetsvektorn, kanske med hjälp av olika privata metoder i klassen!?

Klassen behöver också en metod `collide(Ball/Circle/Turtle b)` som hanterar vad som händer när två objekt krockar.

Om objekt $i$ skall studsa mot objekt $j$ så ges deras nya hastighetsvektorer av formlerna

$$v_i = v_i + \frac{(v_j - v_i) \cdot (s_j - s_i)}{|s_j - s_i|^2}(s_j - s_i)$$

$$v_j = v_j + \frac{(v_i - v_j) \cdot (s_j - s_i)}{|s_j - s_i|^2}(s_j - s_i)$$

där $s_i$ och $s_j$ är bollarnas positionsvektorer. (Operationen $\cdot$ betecknar skalärprodukt.) Bortse från fallet att fler än två objekt kolliderar med varandra samtidigt.

### En klass för lådan

Klassen `Box` håller reda på alla objekten och har en metod förslagsvis med namnet `step` som beordrar objekt att flytta ett steg.

Boxen skall representera lådan som objekten rör sig i och görs då lämpligen som en subklass till `JPanel`.

### En klass för visningsfönstret

Eftersom en `JPanel` inte självständigt kan visas på skärmen måste man lägga in den i en `JFrame`.

### En klass för användargränssnittet

Görs som en subklass till `JFrame`

## Tips på arbetssätt

Generellt: tag små steg! Lägg till och testa en sak i taget!

Samarbeta gärna två och två men alla skall redovisa sin egen kod.

Använd handledarna som bollplank!

Klassen `Vector` skrivs och testas för sig. Den innehåller ingeting utöver vad som ingick i grundkursen.

Därefter kan man skriva och testa klassen `Ball/Circle/Turtle` men tills vidare utan grafiken.

Klassen `Box` kan också implementeras och testas utan grafik.

Det kan var bra att först göra ett körbart program med grafik men utan grafiskt användargränssnitt.

Man kan arbeta med användargränssnittet (det så kallade *GUI*-et) parallellt för att få en känsla för hur detta skall se ut - från början helt utan funktion.