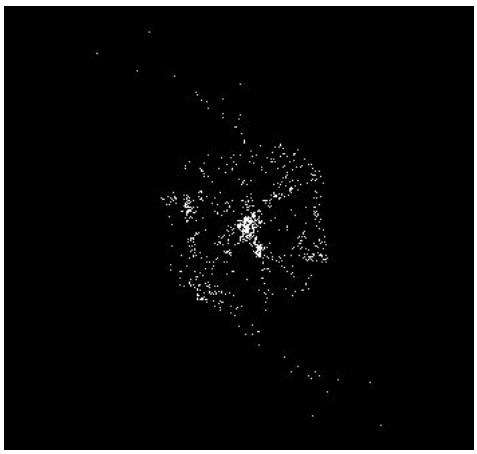
Simulering av stjärnor i en galax

Översikt

I den här uppgifter kommer du att skriva ett program som simulerar hur stjärnor rör sig i en galax. Metoden vi ska använda kallas N-body (som i N antal kroppar) och används i verktyg för att simulera riktiga fysikaliska förlopp. I simuleringen skapar vi först ett antal stjärnor och placerar ut dem i ett koordinatsystem. Därefter börjar vi stega tiden framåt i små steg. I varje tidssteg räknar vi ut åt vilket håll en stjärna dras åt och flyttar den. Detta görs för alla stjärnor vilket exempelvis ger bilden nedan



Figur 1: Uppgiften består av att skriva ett program som får stjärnorna att röra sig enligt Newtons gravitationslag. Med i koden finns en stomme för att rita upp stjärnorna som vita prickar i ett fönster

Uppgiften

Du ska skriva ett program som simulerar stjärnornas rörelser enligt Newtons gravitationslag. Algoritmen som kommer att stakas ut nedan heter N-body och gör att varje stjärna vet i vilken riktning den dras och med vilken kraft.

Programmet ska kunna ta emot två argument

```
foo$ ./galaxy ANTAL_STJÄRNOR ANTAL_ITERATIONER
```

som ger antal stjärnor i simuleringen (`ANTAL_STJÄRNOR`) och antal tidssteg (`ANTAL_ITERATIONER`).

Två stjärnor, A och B, påverkar varandra med en gravitationskraft enligt:

$$F_A = F_B = \frac{M_A * M_B}{r} * G$$

där F_A och F_B är kraften som verkar på A och B, M_A och M_B är deras respektive massa, r är avståndet mellan A och B och G är gravitationskonstanten. Givet kraften, hur räknar vi ut stjärnans nya position? Stjärnans nya x-komponent ges av:

$$a_x = F_X / M_X$$
$$v_x = v_x^{old} + a_x \Delta t$$
$$x = x^{old} + v_x t + \frac{a_x \Delta t^2}{2}$$

y-komponenten räknas ut på motvarande sätt. Δt är tiden (i simuleringstid) som varje tids-loop ska motsvara. Prova att sätta värdet för Δt lågt (~ 0.0001) och öka den så länge simuleringen inte ballar ur.

En stjärna räknar ut summan av alla stjärnors påverkan för att bestämma sin nya koordinat. Detta görs enkelt i två loopar. Nedan exemplifieras dessa två loopar med pseudo-kod.

```
// ett tidssteg
for star i in array_of_stars:
    for star j in array_of_stars:
        if i != j:
            star[i]->ax += compute_force_x(i,j);
            star[i]->ay += compute_force_y(i,j);

update_all_positions()
```

Uppgiften kan sammanfattas i några korta punkter:

- Skapa strukturen för en stjärna med poster för position, acceleration, fart mm.
- Initiera alla stjärnor med en position och en initial fart.
- Skriv tids-loopen, i-loopen, j-loopen och koden som räknar ut den nya kraften.
- Skriv uppdateringsfunktionen.

Kodskelett

I kursrepot finns ett kodskelett för uppgiften. Det finns även ett byggskript med två make-mål (`starsim` och `animate`). Det första målet kompilerar koden; det andra målet kompilerar också koden men bygger även med stöd för att visa en animation av simuleringen ni såg på bilden ovan. Animeringen fungerar på datorer med en X-fönsterserver (t.ex. alla *nix-maskiner på IT). Animeringen har testats på IT-insitutionens maskiner. Du kan prova att köra programmet på insitutionens maskiner och få fönstret skickat över nätet till din lokala dator. Vi visar hur detta går till i vår screencast om att jobba hemifrån med SSH. Se fas0 sprint0 Verktyg 3: SSH.

Animationen kommer att fungera först när det finns stjärnor med x- och y-koordinater att visa.

Bra att veta

Skärmen är förinställd att vara 800 enheter bred och hög (800x800). Centrum för bilden ligger i punkten (400,400) eftersom (0,0) ligger i ett av hörnen. För att få en läcker spridning på stjärnorna kan deras först x och y-värde slumpas som ett tal mellan t.ex. 350 och 450 i både x- och y-riktningen

Förslag till redovisningar kopplade till denna uppgift

Nedan beskriver vi några möjliga kunskapmål vars uppfyllnad kan redovisas som en del av denna uppgift. Listan är inte på något sätt uttömmande. Det är heller inte så att man måste redovisa de kunskapmål som nämns nedan i samband med denna uppgift, eller att denna uppgift är ”den bästa” att redovisa dessa i samband med.

A1 (Procedurell abstraktion) Kodskelettet ger redan viss ledning till hur detta kan göras. Programmet är uppdelat i vettiga funktioner, och för att göra klart uppgiften bör man naturligtvis fortsätta i samma stil. Men vad är en lämplig uppbrytning i funktioner? Finns vissa återkommande mönster som borde brytas ut och kapslas in i en funktion för att undvika upprepningar?

F14 (Rekursiv vs. iterativ) Kodskelettet gör alla beräkningar med s.k. `for`-loopar och rekursion saknas helt (?). Går det att skriva om looparna med rekursion? Vad blir konsekvenserna för koden? Påverkas körtiden?

J27 (Stack och heap) Programmet allokerar redan på både stacken och heapen, men var och varför är det vettigt så som det är skrivet?

O43 (Profilering) Programmet skriver redan ut körtidsdata, så det är enkelt att se hur förändringar av programmet påverkar prestanda. Profilering kan hjälpa till att förklara vilka delar av programmet som körs mest respektive var programmet tillbringar mest tid. Det är alltid intressant att gissa innan man mäter!

R52 (GDB) Acceptera redan nu att du kommer att göra fel. Inse också att det är ett tecken på att du gör framsteg. Programmet kommer att krascha och det kommer inte alltid att vara uppenbart varför. Debuggern kan hjälpa dig att förstå hur programmet faktiskt fungerar, vilka värden variabler har vid vissa tidpunkter, och varför programmet kraschar.