Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Uge(r)seddel 11 - group opgave

Jon Sporring and Christina Lioma

2. – 17. January.
Afleveringsfrist: onsdag d. 18. January kl. 22:00

I denne periode skal I arbejde i grupper. Formålet er at arbejde med:

- Inheritance
- UML diagrams

Opgaverne for denne uge er delt i øve- og afleveringsopgaver.

Øve-opgaverne er:

11ø.0 Simulate a Global Positioning System (GPS) for hikers using object-oriented programming. A GPS for hikers can show the current longitude and latitude (a pair of floating point values). The range of longitude is -180.0 degrees to +180.0 degrees. The range of latitude is -90.0 degrees to +90.0 degrees. The GPS should be able to:

- randomly generate a longitude and latitude pair;
- associate a longitude and latitude pair with a name (e.g. start of hike, hotel, shop). This association (longitude, latitude, name) is called a waypoint. Each waypoint should be saved;
- calculate the distance to a saved waypoint from the current location (randomly generated);
- create a path as a sequence of saved waypoints. Write a GPS class to implement the above, and a separate test class for testing all methods.

Extra challenge: write a method that computes the length of a path, assuming a straight line between each waypoint.

11ø.1 Use object-oriented programming to simulate an online shopping cart that allows customers to:

- add items to it and remove items from it, in different quantities;
- show the total price of all items in the cart at any given time;
- apply a discount to the above total price, if a correct discount code is given by the customer. The amount of the discount can vary, but can be no more than 50%;
- recommend random items to the customer, if the cart is empty for more than 5 minutes;
- cancel and exit the session if the cart is empty for more than 40 minutes;
- save the items in the cart (but not their quantities) as a shopping list;

- add items to the cart from a shopping list (at default quantity of 1).

Write a Cart class to implement the above, and a separate test class for testing all methods. **Extra challenge**: the customer should be allowed to optionally cluster items in the cart in different groups and specify a name for each cluster. Each cluster should be saved. The customer should be able to add or remove the cluster to/from the shopping list.

11ø.2 You are organising a publicity event to launch a new product and you wish to have maximum coverage in the media. To do this, you must select your guests wisely. There is only space to invite 50 guests. Each guest has a name, probability of accepting your invitation, and *media currency* (measured as minutes of media exposure per month). Guests can be divided into the following groups:

- Group 1: these guests have a low probability of accepting your invitation (low probability means 0.1 – 0.4) but very high media currency (500 – 700 minutes per month).
- Group 2: these guests have a probability of accepting your invitation of 0.5 – 0.8, and a media currency of 300 – 400 minutes per month.
- Group 3: these guests have a probability of accepting your invitation of 0.9 – 1.0, and a media currency of 1 – 200 minutes per month.

Produce an object-oriented program that, when instantiating guests from each group, it can randomly assign values to their probability of accepting and to their media currency from the ranges specified in each group. Each guest who attends the event, regardless of group, costs 1000 kr. for catering. In addition, each invitation that is sent out, regardless of whether it is accepted or rejected, costs 200 kr. in administration expenses. Which combination(s) of guests are likely to yield the most press coverage for your event at the lowest possible expense?

11ø.3 For this exercise, you are given a module that produces a graphical simulation of a moving object (the Graph module, available on Canvas). You are asked to translate this module into an object-oriented program by creating two separate classes (one for points, the other for lines). You should test your code and ensure that your OOP version produces the same output as the original module.

11ø.4 Write an object-oriented program that simulates the flight of a cannonball. We wish to find out how far the cannonball will travel when fired at various launch angles and initial velocities. The input to the program will be the launch angle (in degrees), the initial velocity (in meters per second) and the initial height (in meters). The output should be the distance that the projectile travels before striking the ground (in meters).

Let us assume that there are no effects of wind resistance and that the cannonball stays close to earth's surface (i.e., not in orbit). The acceleration of gravity near the earth's surface is about 9.8 meters per second. That means if an object is thrown upward at a speed of 20 meters per second, after one second has passed, its upward speed will have slowed to approximately 20 - 9.8 = 10.2 meters per second. After another second, the speed will be only 0.4 meters per second, and shortly thereafter it will start coming back down.

You should consider the flight of the cannonball in two dimensions: height, so we know when it hits the ground, and distance, to keep track of how far it goes. Think of the position of the cannonball as a point (x, y) in a 2D graph, where the value of y gives the height and the value of x gives the distance from the starting point. Produce a simulation that takes as input the launch angle, the initial velocity, and the initial height, and updates the position of the cannonball to account for its flight.

Some help: Suppose the canonball starts at position $(0, 0)$, and we want to check its position, say, every tenth of a second. In that interval, it will have moved some distance upward (positive y) and some distance forward (positive x). The exact distance in each dimension is determined by its velocity in that direction. Since we are ignoring wind resistance, the x velocity remains constant for the entire flight. However, the y velocity changes over time due to the influence

of gravity. In fact, the y velocity will start out being positive and then become negative as the cannonball starts back down.

Afleveringsopgaven er:

11g.0 I denne opgave skal du implementere en simulator der kan simulere planeternes og Solens bevægelse i vores solsystem. Simulationen består af beregning af Solen og planeternes position og hastighed som funktion af tid. I simulationen inkluderer vi, udover de 8 planeter, også dværgplaneten Pluto.

Modellen Planeternes bevægelse er bestemt af gravitationskraften. Dertil kommer at den enkelte planet følger en banekurve rundt om Solen som har en vis afstand til Solen og de andre planeter, samt har en hastighed som afhænger af hvor på banekurven planeten befinder sig (se Figur 1).

Vi vil antage følgende simplificeringer:

(a) I simulatoren antager vi at planeter er punktformede masser (dvs. de har ingen udbredelse i rummet) og at de ikke roterer om sig selv.

(b) Vi ser desuden bort fra påvirkning fra måner, kometer og andre former for planeter.

(c) Solens masse er langt større end alle andre himmellegemer i solsystemet og derfor vil Solens gravitationstiltrækning på planeterne være betragtelig større end påvirkningen mellem planterne.

(d) Vi kan også antage at planeternes gravitationstiltrækning på Solen er forsvindende lille og derfor sætte gravitationskraften på Solen til nul, dvs. at Solens acceleration er nul, og dermed ændrer Solens hastighedsvektor sig ikke med tiden.

Dette er selvfølgelig grove tilnærmelser til det virkelige solsystem, men modellen kan fungere til forudsigelse af planeternes baner over kortere tidsrum.
Under disse antagelser, kan vi for planet i beregne dens acceleration omkring Solen ved,

$$\mathbf{a}_i = -\frac{GM_{\text{Solen}}}{\|\mathbf{r}_i\|^3}\mathbf{r}_i \tag{1}$$

hvor G er gravitationskonstanten og M_{Solen} er solens masse (se Tabel 1). Vektoren $\mathbf{r}_i$ angiver vektoren fra Solen til planet i, og $\|\mathbf{r}_i\|$ er længden af denne.

Løsningsmetode I simulatoren ønsker vi at beregne de enkelte planeters position $\mathbf{r}(t) = (x(t), y(t), z(t))$ som funktion af tiden t og som resultat af påvirkningen af gravitationsaccelerationen. Tiden t diskretiseres i tidsskridt Δt, således at tiden fremskrives ved hjælp af følgende rekursionsformel

$$t_{n+1} = t_n + \Delta t, \tag{2}$$

og en approximation af vektorerne $\mathbf{r}(t_{n+1})$, og $\mathbf{v}(t_{n+1})$, fremskrives ved[1]

$$\mathbf{r}(t_{n+1}) = \mathbf{r}(t_n) + \mathbf{v}(t_n)\Delta t, \tag{3}$$

$$\mathbf{v}(t_{n+1}) = \mathbf{v}(t_n) + \mathbf{a}(t_n)\Delta t. \tag{4}$$

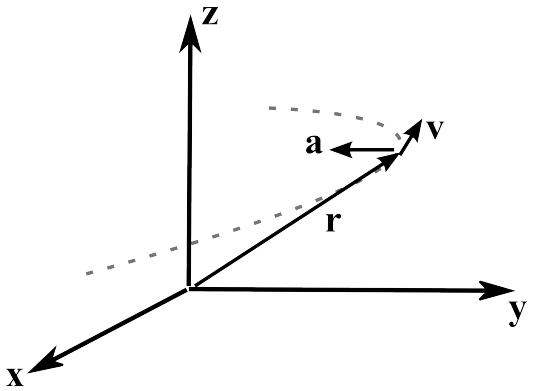
Accelerationen $\mathbf{a}(t_n)$ kan beregnes udfra $\mathbf{r}(t_n)$ for alle planeter ved at anvende (1). Endeligt forudsættes initielle værdier for $\mathbf{r}(t_0)$ og $\mathbf{v}(t_0)$ for hvert himmellegeme kendte.

For at opnå høj præcision i simulatoren er det vigtigt at de initielle værdier angives med høj præcision. Dertil kommer at det er vigtigt at vælge så små tidskridt Δt som muligt, under hensyntagen til at beregningstiden stiger når Δt gøres mindre (flere beregningsskridt er nødvendige for at simulere eksempelvis 1 år frem i tiden).

[1]Disse ligninger er Euler's metode til samtidigt at løse differentialligningerne $\frac{d\mathbf{r}(t)}{dt} = \mathbf{v}(t)$ og $\frac{d\mathbf{v}(t)}{dt} = \mathbf{a}(t)$.

Tabel 1: Relevante naturkonstanter og enheder, samt estimater af produktet mellem gravitationskonstanten og de forskellige planeters masser. Tal fra JPL Ephemeris [1]

Gravitationskonstant, G	6.67384×10^{-11} m$^3/(\text{s}^2 \cdot \text{kg})$
Astronomisk enhed, AU	1 AU = 149597870700 m
GM_{Solen}	$2.959122082322128 \times 10^{-4}$ AU^3/dag^2



Figur 1: En planets tilstand til tiden t angives ved en positionsvektor $\mathbf{r}$, en hastighedsvektor $\mathbf{v}$ og en accelerationsvektor $\mathbf{a}$. Når tiden går, følger planeten en banekurve (illustreret med stiplet kurve) og tilstandsvektorene ændres.

Data For at vurdere hvor præcis din simulation er, skal du sammenligne simulatorens resultat med data fra NASA JPL Ephemeris [1]. JPL Ephemeris er en kombination af observationer af planeternes bevægelse og simulering med høj præcision. NASA anvender blandt andet dette datasæt til planlægning af rummissioner.

Sammen med opgaveteksten finder du en tekstfil for hvert himmellegeme med informationer taget fra JPL Ephemeris datasættet. Datasættet beskriver positionen for de otte planeter og dværgplaneten Pluto ved midnat den første dag i hver måned i perioden 1/1 1966 til 31/12 2016 i et sfærisk koordinatsystem. Formatet filerne er som følger: først kommer der data for planeten, og derefter en tabel af positioner, med en tidspunkt per linje. Den første kolonne er dagens nummer i hhv. Epoch Julian Date (EJD) og, derpå kommer længdegrad (Lon), breddegrad (Lat), og radius (R) for planetens position, som set fra solens centrum. Den sidste kolonne kan ignoreres. EJD er angivet som et reelt tal som tæller antal dage siden en referencedato og kan med fordel anvendes som intern repræsentation af dato-tid i din implementation af simulatoren.

For præcision i beregningerne anvendes ofte astronomiske enheder (AU) i stedet for meter (m). Den astronomiske enhed er defineret ved gennemsnitsafstanden mellem jorden og solen og er fastsat til 1 AU = 149597870700 m. Denne relation kan anvendes til konvertering af tal angivet i meter til astronomisk enhed eller omvendt. I JPL Ephemeris er radius R angivet i enheden AU og vinklerne Lon og Lat i grader.

JPL Ephemeris datasættet benytter det solsystem-barycentriske koordinatsystem, hvor origo er angivet ved solsystemets massecenter og akser angivet udfra jordens orientering på et referencetidspunkt. Du skal anvende dette koordinatsystem i din simulator (dette gøres uden videre ved at anvende JPL Ephemeris data til at angive initielle positioner og hastigheder for planeterne).

Yderligere information om planeterne og Solen kan eksempelvis findes på NASA Planetary Fact Sheets [2], men er ikke nødvendig for at løse denne opgave.

Opgaven I denne opgave skal du designe en solsystem simulator og skrive et program i F#, der implementerer dit design. Minimumskrav til din løsning er at

(a) den kan simulere solsystemets bevægelse over tid;

(b) den kan indlæse JPL Ephemeris data filer for planeterne og solen;

(c) den kan illustrere dine simulationsresultater, og sammenligne dem med JPL Ephemeris data;

(d) brugeren interaktivt kan indtaste, hvor lang tid simulatoren skal simulere;

Opgavens besvarelse skal bestå af

(a) en animation af planeternes bevægelse som beregnet af din simulator i tidsrummet 1/1/2016-31/12/2016. Det skal blot være en 2D animation som viser x og y komponenterne af $\mathbf{r}(t)$;

(b) en tabel som for hver planet og for hvert tidspunkt i NASA JPL Ephemeris's tilhørende datafil viser den euklidiske afstand mellem positionen angivet i NASAs datafil og den beregnet af din simulator.

(c) Et repræsentativt billede fra animationen samt en udskrift af tabellen.

(d) Et UML diagram, som illustrerer dit design.

(e) Skemaer, der beskriver din afprøvning samt udskrift fra kørsel af din unittest.

(f) 1-2 siders tekst i pdf format, som kort beskriver

   i. overvejelser og valg gjort ved udvikling af programmet;

   ii. en kort fortolkning af tabellernes indhold;

   iii. en kort beskrivelse af evt. fejl og mangler ved løsningen.

Ved vurdering af din besvarelse vil der, udover ovenstående, blive lagt vægt på dine valg af datatyper, datastrukturer og dit programs struktur.

Bedømmelseskriterier Der lægges vægt på at der afleveres et kørende program, dvs. at et simpelt og lille men veldesignet, kørende program foretrækkes frem for et ambitiøst eller stort program, der ikke rigtig virker. Det er vigtigt, at individuelle klasser er veldesignet, specificeret, implementeret og afprøvet, selv hvis programmet som helhed ikke kan køre. Det anbefales stærkt at designe, implementere og afprøve et system med meget enkel funktionalitet, inden eventuelle udvidelser føjes til.

Litteratur

[1] NASA Jet Propulsion Laboratory, California Institute of Technology, USA. JPL Ephemeris Horizons System http://ssd.jpl.nasa.gov/horizons.cgi.

[2] NASA, USA. NASA Planetary Fact Sheets http://nssdc.gsfc.nasa.gov/planetary/planetfact.html.