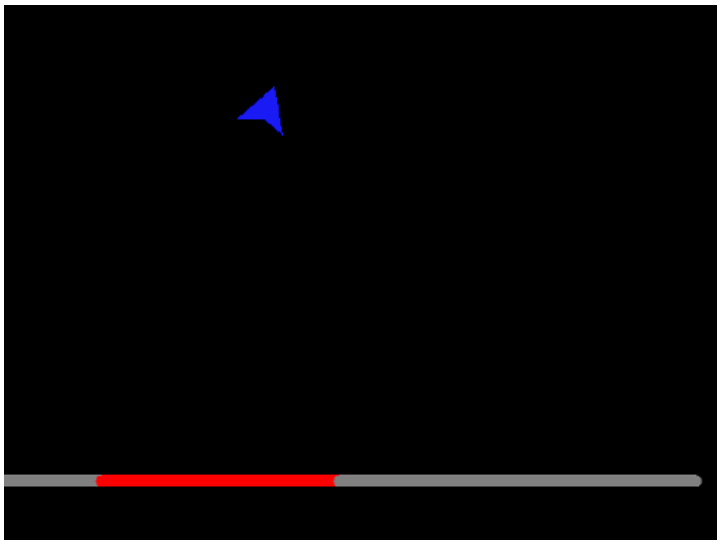
Rosetta og Philae

Erfaren Python PDF

Introduksjon

Romsonden Rosetta ble sendt ut av Den Europeiske Romfartsorganisasjonen (ESA) for å utforske kometen 67P/Tsjurjumov-Gerasimenko. I november 2014 sendte Rosetta ut landingsfartøyet Philae som landet på selve kometen. Vi skal i denne leksjonen bruke skilpaddegrafikk til å lage et enkelt spill hvor målet er å styre Philae slik at det lander perfekt på kometen.



Steg 1: Skilpadden blir et romskip

Vi skal nå bruke skilpadden som om den er et romskip. Vi begynner enkelt med å tegne skilpadden på skjermen.

Sjekkliste

- [ ] Start et nytt Python-prosjekt, og skriv inn følgende kode:

```
import turtle

def tegn_romskip():
    turtle.penup()
    turtle.shapesize(4)
    turtle.setpos(200, 400)
    turtle.setheading(90)
    turtle.color('blue')

tegn_romskip()
input('Trykk Enter')
```

Lagre programmet med navnet `rosetta.py`.

- [ ] Kjør programmet ditt. Skjønner du hva som skjer?

Vi har laget èn funksjon, `tegn_romskip`, som gjør trekanten større, flytter den til toppen av skjermen, roterer den og fargelegger den mørkeblå. Dette vil være romskipet vårt.

- [ ] Linjen `input('Trykk Enter')` har vi lagt til for at ikke turtle-vinduet skal bli borte med en gang. Prøv å ta vekk denne linjen for å se hva som skjer.

- [ ] Vi vil også lage en bakgrunn som ligner litt på en komet. Legg til denne funksjonen før `def tegn_romskip()`:

```
def tegn_bakgrunn():
    turtle.bgcolor('black')
    turtle.speed(11)
    turtle.pensize(10)
    turtle.penup()
    turtle.setposition(-400, -300)
    turtle.pendown()
    turtle.color('grey')
    turtle.forward(800)
```

- [ ] Legg også inn et kall til denne nye funksjonen `tegn_bakgrunn()` rett før `tegn_romskip()` blir kalt.

- [ ] Kjør programmet ditt. Tegnes også en enkel bakgrunn? Endre gjerne litt på de forskjellige tallene og verdiene i koden over, slik at du skjønner hva de forskjellige kommandoene gjør. Kanskje du kan lage en enda bedre bakgrunn?

Steg 2: Romskipet faller mot kometen

La oss begynne med å legge på litt effekt av gravitasjonen, slik at romskipet faller ned mot kometen.

Sjekkliste

- [ ] Først lager vi en variabel som sier hvor sterk gravitasjonseffekten er. Legg denne nesten helt øverst i programmet ditt, rett under `import turtle`, slik at den er lett å finne igjen senere.

```
gravitasjon = -0.01
```

Vi starter med en ganske liten gravitasjonseffekt. Pass på at den er negativ, slik at romskipet vil falle nedover.

- [ ] Vi vil også trenge en beskrivelse av tilstanden til romskipet. I praksis vil dette i denne omgang være hvor fort romskipet beveger seg. Legg til disse linjene rett under definisjonen av `gravitasjon`.

```
romskip = {
    'fart_x': 0.1,
    'fart_y': 0.5
}
```

`romskip` er her en `dict` (`dict` er en forkortelse for *dictionary*, eller *ordbok*). Dette er en veldig nyttig datastruktur i Python, som gjør det lett å samle variabler som hører sammen. Vi kan hente ut verdiene for eksempel ved `romskip['fart_x']`.

- [ ] Vi kan nå skrive koden som flyr romskipet. Legg til denne funksjonen etter definisjonen av `tegn_romskip()`.

```
def fly_romskip():
    while True:
        x, y = turtle.position()
        if y < -270:
            return

        romskip['fart_y'] += gravitasjon
        turtle.setposition(x + romskip['fart_x'], y + romskip['fart_y'])
```

- [ ] Legg også inn et kall til `fly_romskip()` etter kallet til `tegn_romskip()`.

- [ ] Kjør programmet. Faller romskipet ditt ned mot kometen? Stopper romskipet når det kommer til bakken?

Prøv å endre verdiene for `gravitasjon`, `fart_x` og `fart_y`. Hvordan endrer dette bevegelsen til romskipet? Hva gjør tallet `-`

`270` i koden? Hva skjer om du forandrer dette?

- [ ] Den viktigste koden for å flytte romskipet er linjene

```
romskip['fart_y'] += gravitasjon
        turtle.setposition(x + romskip['fart_x'], y + romskip['fart_y'])
```

Skjønner du hvordan disse fungerer? Først endrer vi den vertikale hastigheten (`fart_y`) basert på effekten av gravitasjonen. Deretter flytter vi romskipet så langt som farten tilsier.

Steg 3: Vi styrer romskipet!

Vi skal nå se hvordan vi kan bruke piltastene til å styre romskipet.

Sjekkliste

For at Python skal oppdage at vi trykker på en tast skal vi bruke noe som kalles hendelser (*events* på engelsk). I turtle-biblioteket finnes flere typer hendelser. Vi skal bruke noen som heter `onkey`.

- [ ] La oss begynne enkelt ved å vri romskipet mot høyre når den høyre piltasten trykkes. Lag først en enkel funksjon som snur romskipet mot høyre:

```
def snu_hoyre():
    turtle.right(5)
```

- [ ] Nå må vi fortelle Python at vi vil at denne funksjonen skal kobles til høyre piltast. Legg til disse linjene nederst i `tegn_romskip`-funksjonen:

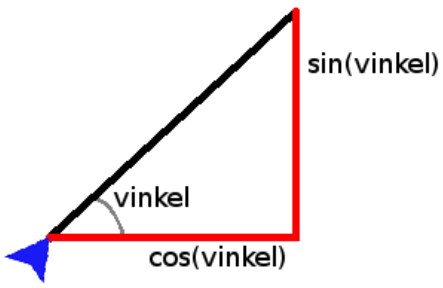
```
turtle.onkey(snu_hoyre, 'Right')
    turtle.listen()
```

Den andre linjen forteller Python at den må lytte etter hendelser.

- [ ] Kjør programmet ditt. Snur romskipet seg mot høyre om du trykker høyre piltast?

- [ ] Skriv nå tilsvarende kode som kan snu romskipet mot venstre. Dette klarer du selv!

Til slutt vil vi bruke pil-opp-tasten for å gi romskipet vårt litt fart. Når vi trykker pil-opp skal romskipet få litt fart i den retningen det peker. Siden vi holder styr på farten i `x`- og `y`-retning separat, må vi fordele denne akselerasjonen. Til dette kan vi bruke matematikk-funksjonene *sinus* og *cosinus*.



- [ ] Sinus og cosinus er en del av `math`-biblioteket i Python, slik at vi må importere dette først. Legg til denne linjen helt øverst, sammen med `import turtle`.

```
import math
```

- [ ] Legg så til en ny funksjon, `bruk_motor`:

```
vinkel = turtle.heading() * math.pi / 180.0
romskip['fart_x'] += math.cos(vinkel)
romskip['fart_y'] += math.sin(vinkel)
```

En liten ekstra komplikasjon i denne koden er at `turtle`- og `math`-bibliotekene regner med vinkler litt forskjellig. `turtle` bruker grader, mens `math` bruker radianer. For å bytte fra grader til radianer har vi ganget vinkelen med pi og delt på 180.

- [ ] Til slutt må vi si at `bruk_motor` skal kalles når vi trykker pil-opp. Legg til denne linjen sammen med de andre `onkey`-linjene:

```
turtle.onkey(bruk_motor, 'Up')
```

- [ ] Prøv spillet ditt igjen. Klarer du å styre romskipet ditt rundt på skjermen? Vær forsiktig slik at du ikke forsvinner i verdensrommet!

Steg 4: Klarer vi å lande trygt?

Avslutningsvis skal vi lage en base på kometen hvor vi vil at romskipet skal lande.

Sjekkliste

- [ ] Vi begynner med å tegne opp et felt hvor vi vil at romskipet skal lande. Bytt ut linjen `turtle.forward(800)` i funksjonen `tegn_bakgrunn` med linjene

```
turtle.forward(200)
turtle.color('red')
turtle.forward(200)
turtle.color('grey')
turtle.forward(400)
```

Om du kjører programmet igjen vil du se en rød base på kometen.

- [ ] Vi vil nå sjekke at landingen er bra. Skriv følgende funksjon,

```
def sjekk_landing():
    x = turtle.xcor()

    if x < -200 or x > 0:
        print('Du landet utenfor basen!')
```

def sjekk_landing():

```
x = turtle.xcor()
vinkel = turtle.heading()                # Ny linje

if x < -200 or x > 0:
    print('Du landet utenfor basen!')
elif abs(vinkel - 90) > 10:              # Ny linje
    print('Du landet skjevt!')           # Ny linje
elif romskip['fart_y'] < -1:             # Ny linje
    print('Du landet for hardt!')        # Ny linje
else:                                    # Ny linje
    print('Perfekt landing!')            # Ny linje
```

- [ ] Prøv spillet ditt igjen. Test at alle de fire forskjellige meldingene virker som de skal!

Prøv selv

Da er vi ferdig med et enkelt kometlander-spill. Den virkelige Rosetta og Philae-operasjonen var nok litt mer komplisert!

Prøv gjerne å videreutvikle spillet ditt. Her er noen ideer:

- [ ] Kan du la utgangsposisjonen og farten til romskipet være tilfeldig? Se på funksjonen `randint()` fra `random`-biblioteket.
- [ ] Kanskje du kan regne ut en poengsum etter landingen? For eksempel basert på hvor hardt man lander, og hvor lang tid man bruker på landingen. For å ta tiden kan du se på funksjonen `time()` i `time`-biblioteket.
- [ ] Kanskje du kan la spilleren automatisk få en ny sjanse dersom han krasjer?
- [ ] Kan du begrense akselerasjonen (se i `bruk_motor`-funksjonen) slik at man ikke så lett forsvinner ut i verdensrommet?

 Forfatter: Geir Arne Hjelle