Battleship!
Avsluttende prosjekt i programmeringskurs
Bjørnegård Skole

Jonas van den Brink

j.v.d.brink@fys.uio.no

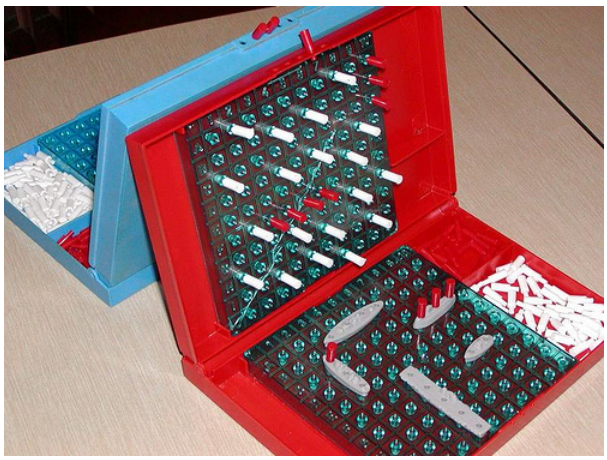
February 9, 2015

Vi har nå kommet til siste uke av kurset, og dere har sett en del Python kode og fått prøvd dere mye selv. Vi skal nå igang med siste del av kurset, der dere selv skal kode en versjon av brettspillet *Battleship!*. Mye av disse oppgaven er hentet fra Code Academy, du kan også løse dem der ved å gå inn på denne lenken og lete deg frem til riktig modul

http://www.codecademy.com/en/tracks/python

Hva er Battleship?

Battleship er et brettspill for to personer. Begge spillerene får utdelt et visst antall skip som de må plassere på et rutenett. Etter begge spillerene har plassert ut alle skipene sine, så gjetter de annenhver gang på koordinatene til mostanderens skip. Etter de gjetter på koordinatene, så svarer motstanderen om det var et treff eller et bom. Spillerene har to brett, et til å holde styr på sine egne skip og et til å holde styr på hvor de har skutt. På figuren under ser du spillbrettet. De hvite pinsene viser skudd som har bommet, og de røde viser skudd som har truffet.



Hva vi skal gjøre i dette prosjektet.

Dette spillet kan dere lage på datamaskin ved hjelp av den programmeringen dere har lært i dette kurset, men vi har litt begrenset tid - så vi skal lage en litt forenklet versjon. For det første holder vi oss til 'single-player', vi starter også med et litt mindre spillbrett og færre skip. Etter du har fått laget et forenklet spill som fungerer bra, så kan du utvide det litt og litt til du har et stort og bra spill! Vi får se hvor langt vi kommer iløpet av denne økten.

La oss sette igang!

Lage spillbrettet

Det første vi må gjøre er å lage selve spillbrettet. Spillbrettet representerer havet og det skal være et rutenett slik at vi kan gjette på en rute ved å gi de to koordinatene, x og y. La oss lage et brett som er 5x5. Hvis vi tegner det for hånd, så hadde det sett sånn her ut:

O	O	O	O	O
O	O	O	O	O
O	O	O	O	O
O	O	O	O	O
O	O	O	O	O

Du kan lage denne på datamaskin ved å bruke en liste med lister. Hvis vi lar en liste bestå av 5 elementer, der hvert element er en liste med 5 elementer som bare er "O" (Stor o, ikke null), så vil det være spillbrettet vårt.

Oppgaver

(a) Lag en variabel, `board`, som er en tom liste.

(b) Legg listen `["O", "O", "O", "O", "O"]` til `board` fem ganger.
Hint: board.append(...).

(c) Skriv ut `board` og se at du har fått totalt 25 ganger "O".

Når du skriver ut `board`, så blir ikke utskriften et fint rutenett sånn som vi tegnet for hånd. La oss fikse dette. Vi kommer til å ha lyst til å skrive ut spillbrettet vårt flere ganger iløpet av spillet, så vi har lyst til å lage en funksjon som gjør dette for oss. Vi kan starte å lage en funksjon som dette

```
def print_board(board):
    ...
```

Husk innrykket! Inne i funksjonen har vi lyst til å ha en løkke som går gjennom `board`-lista og skriver hver rad av spillbrettet på en egen linje.

```
for row in board:
    print row
```

Oppgaver

(d) Lag funksjonen `print_board` ved å fylle inn for '...'

(e) Sjekk at funksjonen fungerer ved å kalle på den med `print_board(board)`.

(f) Istedenfor å bare skrive `print row` inne i for-løkka, kan du skrive

```
print " ".join(row)
```

Da blir utskriften enda finere.

Før du går videre bør du få en utskrift som er et fint rutenett, hvis du ikke får til det er det på tide å få litt hjelp, enten fra sidemannen eller fra oss.

Nå som du har laget spillbrettet vi skal bruke og du kan skrive det ut pent, må du forstå hvordan vi kan endre på spillbrettet. Jeg minner nå om at vi endrer på en liste ved hjelp av *indeksering*. Hvis vi har en enkel liste

```
names = ["Lisa", "Mary", "Marcus"]
```

så kan vi for eksempel endre 'Marcus' til 'Markus' ved å skrive

```
names[2] = "Markus"
```

Husk at python begynner å telle på 0!

Siden spillbrettet vårt består av lister inne i lister, så vil for eksempel `board[0]` være listen `["O", "O", "O", "O", "O"]` som vi da kan indeksere på nytt! Altså kan vi gjøre for eksempel dette

```
board[0][0] = "X"
```

Vi har nå endret koordinat (0, 0) på spillbrettet vårt til å være 'X', istedenfor 'O'. Prøv deg litt frem til du har god forståelse av hvordan det fungerer

Oppgaver

(g) Kjør kommandoen `board[0][0] = "X"` og bruk deretter `print_board(board)` for å se resultatet. Hvilket element endret vi på?

(h) Endre feltet i midten av spillbrettet til å være 'X'.

Nå har vi altså laget spillbrettet vårt, laget en funksjon `print_board` som skriver det ut som et fint rutenett og vi har funnet ut hvordan vi kan gå inn og endre det. La oss nå se på hvordan vi kan plassere ut og skyte på skipene.

Plassere ut skip

La oss starte med å plassere ut et fiendtlig skip som bare dekker 1 enkelt rute til å begynne med. Skipet skal kunne være hvor som helst på hele kartet. Siden kartet vårt er et 5x5 rutenett, kan vi si at ruten skipet er i vil ha en x og en y-koordinat som går fra 0 til 4.

Det skal være tilfeldig hvor skipet er hver gang spillet spilles. Du kan få til å plassere skipet tilfeldig ved å trekke koordinatene tilfeldig.

Oppgaver

(i) Importer funksjonen `randint` fra random.

(j) Lag en variabel `ship_x` og sett den lik et tilfeldig tall mellom 0 og 4.

(k) Lag en variabel `ship_y` og sett den lik et tilfeldig tall mellom 0 og 4.

Vi har nå trukket koordinatene til skipet tilfeldig og lagret dem i variabelene `ship_x` og `ship_y`. For koden vi skal skrive videre er det nyttig å vite akkurat hvor skipet befinner seg, sånn at vi kan teste programmet vårt lettere. Du kan finne ut hvor skipet er rett og slett ved enten å skrive ut koordinatene

```
print ship_x
print ship_y
```

eller du kan endre spillbrettet så man ser det når man skriver ut brettet

```
board[ship_x][ship_y] = "+"
print_board(board)
```

Uansett hvilken av disse løsningene du går for, er det viktig at du husker å fjerne utskriften når du har skrevet og testet ferdig programmet ditt, ellers blir spillet altfor lett for den som spiller det.

Skyte på skipet!

Vi er nå klare for å få brukeren til å prøve å skyte ned skipet, da må vi få brukeren til å gjette på hvor skipet er. Forhåpentligvis er du vant med hvordan du stiller brukeren spørsmål, det gjør du med `raw_input()`.

(l) Lag en variabel `guess_x` og bruk `raw_input()` til å lagre svaret fra brukeren.

(l) Funksjonen `raw_input()` vil alltid lage tekststrenger, men vi vil at `guess_x` skal være et heltall. Gjør slik at `guess_x` er et heltall.
Hint: `guess_x = int(raw_input(...))`

(m) Lag en variabel `guess_y` og gjør tilsvarende som for x-koordinaten.

Nå som vi har trukket posisjonen til skipet tilfeldig, og vi har spurt brukeren hvor de har lyst til å skyte, så er vi klare til å sjekke om brukeren traff. Dette gjør vi selvsagt med en `if`-test. I første omgang er det bare to muligheter, enten traff brukeren, eller så bommet hun. For at det skal være et treff, så må selvfølgelig begge koordinatene stemme! Altså må `ship_x` og `guess_x` være like **og** `ship_y` og `guess_y` være like.

(n) Skriv en `if`-test som sjekker om `ship_x == guess_x` og `ship_y == guess_y`. Hvis det er tilfellet, skriv ut en beskjed om at brukeren traff.

(o) Legg til en `else`-blokk der du skriver ut en beskjed om at brukeren bommet.

(p) Test programmet ditt, der du med vilje treffer. Prøv så med en bom. Fungerer alt som forventet?

I tilegg til å bare skrive ut en beskjed dersom brukeren gjetter feil, bør vi endre spillbrettet så brukeren kan se hvor de allerede har skutt. La oss bruke 'X' for å vise at et felt har blitt skutt på men var et bom.

(q) I `else`-blokken din, endre feltet i spillbrettet som har koordinater `guess_x` og `guess_y` til `"X"`. Skriv også ut brettet
Hint:

```
if ...:
    print ...
else:
    print ...
    board[...][...] = ...
    print_board(board)
```

(p) Test programmet ditt med en bom for å se at det fungerer som det skal.

Flere skudd

Hitil har brukeren bare fått lov til å prøve å skyte ned skipet 1 gang, det er litt lite! Vi skal nå bruke en løkke sånn at brukeren kan skyte opp til 10 ganger! Vi velger å bruke en `while`-løkke som gjentar seg helt til brukeren har enten vunnet, eller bommet 10 ganger. Vi må da holde styr på hvor mange ganger brukeren har bommet.

(r) Lag en variabel `misses` og sett den til å være 0.

(s) Gi all kode som skal gjenta seg selv et innrykk, du gjør dette enkelt ved å merke alle linjer som skal ha innrykk samtidig og trykke på tab-knappen.
Hint: Det vi vil at skal gjenta seg selv, er at brukeren gjetter på koordinater, og så at vi sjekker gjettene og oppdaterer brettet

(t) Rett før kodeblokken du nettopp ga innrykk, start `while`-løkken din.
Hint: `while misses ... :`

(u) Inne i løkka di må du huske på å øke variabelen `misses` når brukeren bommer. Oppdater programmet ditt slik at den teller opp hver gang brukeren bommer.
Hint: `misses += 1`

(v) På starten av løkka di, før du spør brukeren om å gjette. Skriv ut en beskjed som forteller hvor mange skudd de har igjen.

(w) Test programmet ditt og sjekk at løkka fungerer som den skal.

En ting som er litt uheldig for brukeren nå, er de ved uhell kan skyte på samme felt som de allerede har prøvd. La oss endre programmet vårt slik at brukeren ikke bruker opp et forsøk hvis de gjør dette. Vi må da gå inn i `else`-blokken vår og legge inn en ny test, som sjekker om `board[guess_x][guess_y]` er `"X"`. Hvis det er tilfellet, så har brukeren allerede skutt på det feltet!

(w) På starten av `else`-blokken din, legg inn en ny test, som sjekker om `board[guess_x][guess_y]` er `"X"`, hvis det er tilfellet, skriv en beskjed til brukeren om at de allerede har skutt på det feltet. Legg koden som allerede hørte til `else`-blokken under en ny `else`-blokk.
Hint:

```
if guess_x == ship_x ... :
    # User hit the target!
else:
    if board[guess_x][guess_y]...:
        # User tried firing on a coordinate they already tried!
    else:
        # User misses!
```

Game over!

Nå begynner spillet ditt og bli ganske så bra! Et problem derimot, er at spillet ikke er ferdig når brukeren gjetter riktig! Det er heller ingen beskjed til brukeren hvis de ikke greier å treffe skipet med sine 10 skudd. La oss fikse begge disse to problemene.

Først, hvis brukeren gjetter riktig, så må vi sørge for at løkka ikke gjentar seg. Dette kan vi gjøre med kommandoen `break`. Når python ser en `break`-kommando, så hopper den umiddelbart til den første kodelinjen etter hele løkka, så vi tar rett og slett å bryter løkka.

(x) Legg inn en break etter at vinner-beskjeden er skrevet ut.

Nå vil vi at det skal skrives ut en beskjed til brukeren at de tapte hvis de ikke traff på noen av sine 10 skudd. Dette kan vi gjøre med nok en `if`-test. La oss legge den etter løkka er ferdig å kjøre. Hvis løkka er ferdig fordi den ble brytet med `break`, så vil `misses` være mindre enn 10. Hvis løkka er ferdig fordi brukeren har bommet 10 ganger, så er `misses` 10.

(y) Legg in en `if`-test etter `while`-løkka som sjekker om `misses` er 10. Hvis det er sant, skriv ut en beskjed til brukeren om at de har tapt.

Nå gjenstår det bare å teste programmet og finne alle mulige småfeil som kan være igjen!

(z) Test programmet ditt til du er fornøyd med det. Få så en venn til å prøve seg!

Du har nå laget et komplett Battleship!-spill. Bra jobba! Det som følger nå er måter man kan utvide spillet hvis man har lyst til det.

Her er en liste med forslag for utvidelser man kan gjøre

- Lag en fin introduksjonsmelding.
- Legg inn flere tester, hvis for eksempel brukeren skyter utenfor spillbrettet bør de få beskjed om det!
- Legg inn flere spillmoduser. Kanskje for eksempel vansklighetsgrader med forskjellige størrelser på spillbrettet eller forskjellige antall skudd. Brukeren kan velge modus fra en meny på starten.
- Legg ut flere skip på brettet! Brukeren må treffe alle for å vinne. Pass på å ikke legge skip over hverandre!
- Legg ut skip som dekker flere felt, denne er litt utfordrende, men kan være stilig!
- Gi spilleren spesialmisilir som man bare kan bruke 1 gang. Kanskje den treffer et kors (5 felter) på en gang. Eller tar ut en hel rad/kolonne av spillbrettet!