Python

Tre på rad mot datamaskina

Skrevet av: Omsett frå Code Club UK (//codeclub.org.uk)

Oversatt av: Stein Olav Romslo

Kurs: Python
Tema: Tekstbasert, Spill
Fag: Programmering
Klassetrinn: 8.-10. klasse

Introduksjon

Denne gongen skal me prøve å skrive kode slik at datamaskina kan spele tre på rad mot oss. Datamaskina vil ikkje spele så bra i starten, men etter kvart som den lærer nokre triks kan den kanskje klare å vinne mot deg!

Steg 1: Me fortset frå førre gong

I tre på rad (../tre_pa_rad/tre_pa_rad_nn.html) laga me eit tre på rad-spel for to spelarar. Me brukte *Tk-lerretet* frå `tkinter` -biblioteket for å teikne på skjermen. La oss sjå på kva me allereie har før me startar å skrive ny kode.

Sjekkliste

- [ ] Åpne IDLE. Åpne fila frå førre gong og lagre den med eit nytt namn. Viss du ikkje finn att fila kan du kopiere inn følgjande:

```
from tkinter import *

main = Tk()

c = Canvas(main, width=600, height=600)
c.pack()

c.create_line(200, 0, 200, 600)
c.create_line(400, 0, 400, 600)

c.create_line(0, 200, 600, 200)
c.create_line(0, 400, 600, 400)

grid = [
    "0", "1", "2",
    "3", "4", "5",
    "6", "7", "8",
]

def click(event):
    shape = choose_shape()
    across = int(c.canvasx(event.x) / 200)
    down = int(c.canvasy(event.y) / 200)
    square = across + (down * 3)

    if grid[square] == "X" or grid[square] == "O":
        return

    if winner():
        return

    if shape == "O":
        c.create_oval(across * 200, down * 200,
            (across+1) * 200, (down+1) * 200)
        grid[square] = "O"
    else:
        c.create_line(across * 200, down * 200,
            (across+1) * 200, (down+1) * 200)
        c.create_line(across * 200, (down+1) * 200,
            (across+1) * 200, down * 200)
        grid[square] = "X"

def choose_shape():
    if grid.count("O") > grid.count("X"):
        return "X"
    else:
        return "O"
```

```
def winner():
    for across in range(3):
        row = across * 3
        line = grid[row] + grid[row+1] + grid[row+2]
        if line == "XXX" or line == "OOO":
            return True

    for down in range(3):
        line = grid[down] + grid[down+3] + grid[down+6]
        if line == "XXX" or line == "OOO":
            return True

    line = grid[0] + grid[4] + grid[8]
    if line == "XXX" or line == "OOO":
            return True

    line = grid[2] + grid[4] + grid[6]
    if line == "XXX" or line == "OOO":
            return True

c.bind("<Button-1>", click)

mainloop()
```

- [ ] Lagre og køyr programmet, slik at du er sikker på at det virkar!

  Du skal kunne klikke i rutene for å plassere sirklar og kryss inntil nokon får tre på rad.

- [ ] Før me startar med dagens kode vil me gjere litt opprydding i koden for at me enklare skal kunne lese kva som skjer i prosedyra `click`. Me flyttar koden som teiknar sirklar og kryss til ei eiga prosedyre. Bytt ut prosedyra `click` med desse to prosedyrene:

```
def click(event):
    shape = choose_shape()
    across = int(c.canvasx(event.x) / 200)
    down = int(c.canvasy(event.y) / 200)
    square = across + (down * 3)

    if grid[square] == "X" or grid[square] == "0":
        return

    if winner():
        return

    grid[square] = shape
    draw_shape(shape, across, down)

def draw_shape(shape, across, down):
    if shape == "0":
        c.create_oval(across * 200, down * 200,
            (across+1) * 200, (down+1) * 200)
    else:
        c.create_line(across * 200, down * 200,
            (across+1) * 200, (down+1) * 200)
        c.create_line(across * 200, (down+1) * 200,
            (across+1) * 200, down * 200)
```

Køyr koden og test at den framleis fungerer på same måte som før. Dette er eit døme på noko som kallast *refaktorering*. Me har endra på sjølve koden, men ikkje endra korleis programmet fungerer.

Steg 2: Spel tilfeldig

Før me kan lære datamaskina korleis den gjer gode trekk må me lære den korleis den gjer trekk i det heile. Me startar med å la datamaskina finne ei tilfeldig ledig rute, og så spele der.

Hugs at me har ein variabel som heiter `grid` som kan fortelje oss korleis brettet ser ut. Det er ei liste som startar som `["0", "1", "2", ... ]`, der me set inn `"X"` og `"0"` etter kvart som me speler. Me startar med å finne ledige ruter i denne lista, for så å spele ei slik rute.

Sjekkliste

- [ ] Fyrst vil me lage ei ny prosedyre, `free_squares`, som kan finne ledige ruter. Legg til denne koden under prosedyra `winner`, men over linja `c.bind(...)`:

```
def free_squares():
    output = []
    for position, square in enumerate(grid):
        if square != "X" and square != "O":
            output.append(position)
    return output
```

Denne prosedyra lagar ei tom liste. Så går den gjennom heile rutenettet og sjekkar kvar rute om ho er tom.

Kommandoen `enumerate` kan fortelje oss posisjonen til kvart element i `grid`-lista. Til dømes vil `enumerate` gjere om ei liste `['A','B','C']` til para `(0, 'A'), (1,'B'), (2, 'C')` slik at me ikkje trenger å telje elementa sjølv.

- [ ] På toppen av fila vil me importere `random`-biblioteket, som me vil bruke for å tilfeldig velje eit trekk

```
from tkinter import *
import random
```

Du hugsar kanskje at me brukte `random.choice` i ei anna oppgåve om Hangman.

- [ ] No skriv me ei prosedyre `play_move()` som kan spele i ei tilfeldig tom rute. Legg til denne prosedyra etter `free_squares`, men før linja `c.bind(...)`

```
def play_move():
    moves = free_squares()
    square = random.choice(moves)

    across = square % 3
    down = square // 3

    grid[square] = "X"
    draw_shape("X", across, down)
```

Fyrst brukar me `free_squares` til å lage ei liste over dei tomme rutene. Så vel me ei tilfeldig av desse rutene. No vil me omsetje dette rutenummeret til rad- og kolonnenummer. Dette gjer me ved å bruke `%`- og `//`-operatorane. La oss sjå litt nærare på korleis dette virkar:

```
    0 1 2
    -----
0 | 0 1 2
1 | 3 4 5
2 | 6 7 8
```

Til dømes er rute nummer 5 i rad 1 og kolonne 2. Viss me deler 5 på 3 får me 1 med 2 i reist.

`5 // 3` er 1, `6 // 3` er 2, og så bortetter. Operatoren `//` fortel oss kor mange gonger eit tal deler eit anna, men ser bort frå reisten. Sidan me har 3 kolonner fortel `5 // 3` oss i kva rad rute 5 er.

`5 % 3` er 2, `6 % 3` er 0. Operatoren `%` fortel oss kva reisten er når me deler eit tal med eit anna. Dette gir oss kolonnenummeret.

Legg merke til at dei to linjene

```
across = square % 3
down = square // 3
```

gjer den motsette utrekninga av

```
square = across + (down * 3)
```

som me allereie har brukt i `click`.

- [ ] Til slutt endrar me `click`-prosedyra slik at den kallar `play_move`. På denne måten vil fyrst spelaren gjere sitt trekk, og så gjer datamaskina sitt trekk.

```
def click(event):
    across = int(c.canvasx(event.x) / 200)
    down = int(c.canvasy(event.y) / 200)
    square = across + (down * 3)

    if grid[square] == "X" or grid[square] == "O":
        return

    if winner():
        return

    grid[square] = "O"
    draw_shape("O", across, down)

    if winner():
        return

    play_move()
```

Fyrst sjekkar me om spelaren har vunne, og viss ikkje let me datamaskina gjere sitt trekk.

- [ ] Lagre programmet og køyr det. No vil datamaskina trekke etter deg. Den vil ikkje spele spesielt bra sidan den berre gjer tilfeldige trekk.

Steg 3: Vel eit trekk som vinn

No speler datamaskina tre på rad, men den er ikkje spesielt flink. La oss hjelpe den litt. I staden for å velje trekk heilt tilfeldig, så skal datamaskina velje trekk som gjer at den vinn viss det finst eit slikt trekk. Ideen er at me kan sjekke alle dei moglege trekka til datamaskina, og viss eitt av desse vil vinne spelet let me datamaskina spele det.

Sjekkliste

- [ ] Endre prosedyra `winner` slik at den tek eit argument `grid`:

```
def winner(grid):
    for across in range(3):
        row = across * 3
        line = grid[row] + grid[row+1] + grid[row+2]
        if line == "XXX" or line == "OOO":
            return True

    for down in range(3):
        line = grid[down] + grid[down+3] + grid[down+6]
        if line == "XXX" or line == "OOO":
            return True

    line = grid[0] + grid[4] + grid[8]
    if line == "XXX" or line == "OOO":
            return True

    line = grid[2] + grid[4] + grid[6]
    if line == "XXX" or line == "OOO":
            return True
```

Du treng berre å endre den fyrste linja i prosedyra. Dette tyder at `winner` vil bruke ei liste me sender til den, i staden for `grid` som hugsar korleis dette spelet ser ut. Dermed kan `winner` òg undersøke trekk som ikkje har blitt spelt endå.

- [ ] No må me forandre `click` så den sender inn riktig liste.

```
def click(event):
     across = int(c.canvasx(event.x) / 200)
    down = int(c.canvasy(event.y) / 200)
    square = across + (down * 3)

    if grid[square] == "X" or grid[square] == "O":
        return

    if winner(grid):
        return

    grid[square] = "O"
    draw_shape("O", across, down)

    if winner(grid):
        return

    play_move()
```

Alle stader me har `winner()` i koden byter me det ut med `winner(grid)`.

- [ ] Køyr koden, den skal framleis virke akkurat som før, for me har framleis ikkje endra korleis datamaskina speler.

- [ ] La oss hjelpe datamaskina ved å leggje til nokre linjer i `play_move` som kan leite etter vinnande trekk!

```
def play_move():
    moves = free_squares()
    square = random.choice(moves)

    # Bruk eit vinnande trekk viss det eksisterer
    for possible in moves:
        new_grid = list(grid)
        new_grid[possible] = "X"
        if winner(new_grid):
            square = possible
            break

    across = square % 3
    down = square // 3

    grid[square] = "X"
    draw_shape("X", across, down)
```

For kvar ledige rute lagar me ein kopi av `grid` -lista med kommandoen `list(grid)` . Så plasserer me ein X i den ledige ruta og brukar `winner` for å undersøke om dette vil vere eit vinnande trekk!

- [ ] Køyr programmet ditt og test det fleire gonger. Datamaskina skal ha blitt litt flinkare til å spele no.

Steg 4: Vel eit trekk som blokkerer

Den andre strategien me vil lære datamaskina er å blokkere trekk som gjer at me vil vinne. Det gjer me på nesten same måte, men no ser me kva som skjer viss me plasserer O i dei ledige rutene.

Sjekkliste

- [ ] Legg til litt meir kode i `play_move` som blokkerer trekk som gjer at spelaren kan vinne.

```
def play_move():
    moves = free_squares()
    square = random.choice(moves)

    # Bruk eit blokkerande trekk viss det eksisterer
    for possible in moves:
        new_grid = list(grid)
        new_grid[possible] = "O"
        if winner(new_grid):
            square = possible
            break

    # Bruk eit vinnande trekk viss det eksisterer
    for possible in moves:
        new_grid = list(grid)
        new_grid[possible] = "X"
        if winner(new_grid):
            square = possible
            break

    across = square % 3
    down = square // 3

    grid[square] = "X"
    draw_shape("X", across, down)
```

Legg merke til at datamaskina fyrst plukkar ei tilfeldig rute. Så sjekkar den om den kan blokkere, og viss den kan det, så ombestemmer den seg. Til slutt sjekkar den om den kan vinne, og viss den kan det, så ombestemmer den seg. Til slutt sjekkar den om den kan vinne, og viss den kan det så ombestemmer den seg ein gong til!

- [ ] Køyr koden og sjå om du klarar å vinne mot datamaskina! Det har no blitt mykje vanskeligere.

Heile programmet

Det ferdige programmet ditt vil sjå ut omlag som dette!

```
from tkinter import *
import random

main = Tk()

c = Canvas(main, width=600, height=600)
c.pack()

c.create_line(200, 0, 200, 600)
c.create_line(400, 0, 400, 600)

c.create_line(0, 200, 600, 200)
c.create_line(0, 400, 600, 400)

grid = [
    "0", "1", "2",
    "3", "4", "5",
    "6", "7", "8",
]

def click(event):
    across = int(c.canvasx(event.x) / 200)
    down = int(c.canvasy(event.y) / 200)
    square = across + (down * 3)

    if grid[square] == "X" or grid[square] == "O":
        return

    if winner(grid):
        return

    grid[square] = "O"
    draw_shape("O", across, down)

    if winner(grid):
        return

    play_move()

def draw_shape(shape, across, down):
    if shape == "O":
        c.create_oval(across * 200, down * 200,
            (across+1) * 200, (down+1) * 200)
    else:
        c.create_line(across * 200, down * 200,
            (across+1) * 200, (down+1) * 200)
        c.create_line(across * 200, (down+1) * 200,
            (across+1) * 200, down * 200)
```

```
def winner(grid):
    for across in range(3):
        row = across * 3
        line = grid[row] + grid[row+1] + grid[row+2]
        if line == "XXX" or line == "OOO":
            return True

    for down in range(3):
        line = grid[down] + grid[down+3] + grid[down+6]
        if line == "XXX" or line == "OOO":
            return True

    line = grid[0] + grid[4] + grid[8]
    if line == "XXX" or line == "OOO":
        return True

    line = grid[2] + grid[4] + grid[6]
    if line == "XXX" or line == "OOO":
        return True

def free_squares():
    output = []
    for position, square in enumerate(grid):
        if square != "X" and square != "O":
            output.append(position)
    return output

def play_move():
    moves = free_squares()
    square = random.choice(moves)

    # Bruk eit blokkerande trekk viss det eksisterer
    for possible in moves:
        new_grid = list(grid)
        new_grid[possible] = "O"
        if winner(new_grid):
            square = possible
            break

    # Bruk eit vinnande trekk viss det eksisterer
    for possible in moves:
        new_grid = list(grid)
        new_grid[possible] = "X"
        if winner(new_grid):
            square = possible
            break
```

```
    down = square // 3
    across = square % 3

    grid[square] = "X"
    draw_shape("X", across, down)

c.bind("<Button-1>", click)

mainloop()
```

Utfordring

Det er framleis mogleg å vinne mot datamaskina. Kan du gjere endringar som gjer at den speler endå betre? Kanskje du kan lære datamaskina å spele perfekt?