PGZ - Sprettball

Ekspert Python PDF

Introduksjon

I denne oppgaven skal du lage en ballanimasjon, ved hjelp av det du har lært i oppgaven om enkle objekter, samt Pygame og Pygame Zero. Dersom du ikke husker objekter, kan du gå tilbake og raskt repetere.

Steg 1: Høyde og bredde

Lag et nytt python-program med følgende kode:

```
HEIGHT = 400
WIDTH = 600
```

Kjør programmet, og se hva som skjer. Du skal nå se et svart vindu som er 400 piksler høyt, og 600 piksler bredt.

En **piksel** er et lyspunkt på skjermen og nøyaktig hvor stort dette lyspunktet er avhenger av hvilken skjerm du har - dermed kan det være at vinduet får ulik størrelse på andre datamaskiner enn din egen.

Steg 2: Lag en ball!

Vi skal nå lage en ball som vi kan vise på skjermen. Vi begynner med å lage en `Ball`-klasse, som har variablene `radius` og `color`, samt en posisjon bestående av `x` og `y`.

```
COLORS = {
   'red': (255, 0, 0),
   'green': (0, 255, 0),
   'blue': (0, 0, 255),
   'white': (255, 255, 255),
   'black': (0, 0, 0)
}

class Ball:
   radius =  20
   color = COLORS['red']
   x = WIDTH // 2
   y = HEIGHT // 2
```

Vi har her valgt å ha en rød ball, men du kan velge en annen farge fra `COLORS`-ordboka om du vil det. Husk at `//` betyr 'heltallsdivisjon', dvs at svaret rundes av nedover, slik at vi får et helt tall som svar.

Vi må i tillegg ha en funksjon som kan tegne ballen vår. Denne skal vi kalle for `draw()`. Husk på at funksjonene som skal være en del av klassen må ha et innrykk. Vi må dermed endre på klassen, slik at den ser slik ut:

```
class Ball:
   radius = 20
   color = COLORS['red']
   x = WIDTH // 2
   y = HEIGHT // 2

   def draw(self):
      screen.draw.filled_circle((self.x, self.y), self.radius, self.color)
```

Nå må er du nesten ferdig. Vi må lage et `Ball`-objekt, `ball1` og en global `draw`-funksjon. Dette vil se slik ut:

```
ball1 = Ball()

def draw():
    screen.clear()
    ball1.draw()
```

`screen.clear()` sørger for at vi tegner på en blank skjerm, og må alltid komme først i den globale funksjonen `draw()`.

Test programmet ditt

Du kan nå teste programmet ditt. Du skal få opp en ensfarget sirkel midt i vinduet.

Steg 3: Bevegelse

Vi vil at ballen vår skal bevege seg. Hvordan skal vi få til dette? Vi lager funksjonen `update()`.

Først må vi legge til et par variabler som bestemmer farten på ballen. Vi skal her ha en variabel for farten i y-retning, og en variabel for farten i x-retning.

```
class Ball:
    radius = 20
    color = COLORS['red']
    x = WIDTH // 2
    y = HEIGHT // 2
    speed_x = 3
    speed_y = 3
```

Så må vi lage en en funksjon `update()` som er en del av `Ball`. Denne sørger for at ballen beveger seg `speed_x` piksler i x-retningen, og `speed_y` i y-retningen.

```
class Ball:
    # ...

    def update(self):
        self.x += self.speed_x
        self.y += self.speed_y
```

I tillegg må vi ha en global funksjon `update()` som kaller `ball1.update()`:

```
def update():
    ball1.update()
```

Test programmet ditt

Du kan nå teste programmet ditt igjen. Ballen skal nå bevege seg, dersom alt er gjort riktig.

Hva skjer når den kommer til kanten? I neste steg skal vi sørge for at ballen ikke forsvinner ut av vinduet.

Steg 4: Veggkollisjoner

Vi ønsker å la ballen sprette tilbake når den treffer en vegg. Her er det et par ting vi må tenke på - hvordan oppdager vi at ballen treffer veggen, og hvordan kan vi endre variablene slik at den spretter vekk fra veggen? Ballens posisjon bestemmes av `x` og `y` men den har også `radius` som vi må ta hensyn til når vi skal oppdage om ballen treffer veggen. Når ballen treffer den øverste eller den nederste veggen ønsker vi at farten reverseres i y-retning, det samme gjelder for farten i x-retning når vi treffer høyre eller venstre vegg.

Vi må endre `update()`-funksjonen i `Ball`-klassen:

```
class Ball:
    # ...

    def update(self):
        self.x += self.speed_x
        self.y += self.speed_y

        # sjekker for kollisjon i x-retning
        if self.x + self.radius >= WIDTH or self.x - self.radius <= 0:
            self.speed_x = -self.speed_x

        # sjekker for kollisjon i y-retning
        if self.y + self.radius >= HEIGHT or self.y - self.radius <= 0:
            self.speed_y = -self.speed_y
```

Test programmet ditt

Kjør programmet ditt, og pass på at ballen spretter tilbake når den treffer en av veggene.

Steg 5: Styre farta til ballen

Vi skal la brukeren styre farta til ballen ved hjelp av piltastene. Når brukeren trykker på 'Pil opp' skal ballen gå raskere oppover (evt. mindre fort nedover), det motsatte skal skje om brukeren trykker 'Pil ned'. Det samme skal skje om brukeren trykker på 'Pil høyre' eller 'Pil venstre', men da skal fartsendringa skje i x-retning.

For å få til dette skal vi lage en `on_key_down()`-funksjon i `Ball`-klassen:

```
class Ball:
    # ...

    def on_key_down(self, key):
        if key == keys.LEFT:
            self.speed_x -= 1
        elif key == keys.RIGHT:
            self.speed_x += 1
        elif key == keys.UP:
            self.speed_y -= 1
        elif key == keys.DOWN:
            self.speed_y += 1
```

Legg merke til at funksjonen har et parameter, `key`, som brukes til å avgjøre hvilken tast brukeren trykket på.

Vi trenger også en global `on_key_down()`-funksjon. Denne har også en `key`-parameter, som sendes videre til `ball1.on_key_down()`.

```
def on_key_down(key):
    ball1.on_key_down(key)
```

Test programmet ditt

Du skal nå ha en ball som spretter mellom vinduskantene, og du skal kunne styre farten ved hjelp av piltastene.

Utfordring: Stopp ballen

Vi ønsker å bruke mellomromstasten for å stoppe ballen. Dvs. sette `speed_x` og `speed_y` til `0`. Prøv å endre funksjonen `on_key_down(key)` i `Ball`-klassen for å sjekke om brukeren har trykket på mellomromstasten.

Hint: `key == keys.SPACE` vil være sant dersom brukeren trykker på mellomromstasten.