RAPPORT OPPGAVE 2

INF-1400-Objektorientert programmering

22. februar 2019

Martin Soria Røvang
Universitetet i Tromsø

Inneholder 8 sider, inkludert forside.

Innhold

1 Introduksjon

1.1 Krav

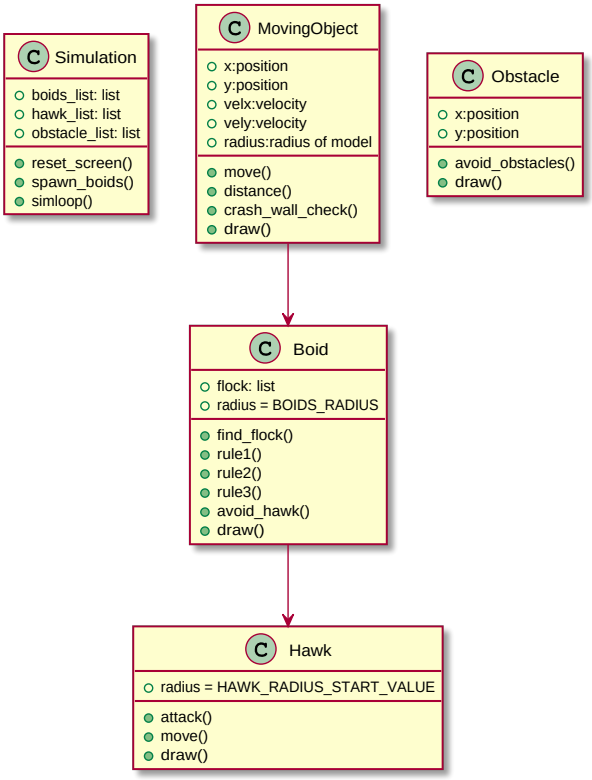
2 Teknisk bakgrunn

ARV
super

3 Design

3.1 Klassestruktur

I dette prosjektet har vi klassestrukturen som vist i figur(1) under,



Figur 1: Klassestrukturen i programmet.

Her arver *Boid* klassen fra *MovingObject* og *Hawk*(Hoik) fra Boid. Hvis man arver fra en generell struktur kan man spare kodeplass og generelt får mer strukturert kode. Hvis man arver fra en klasse får man alle metodene og attributtene fra klassen man arver fra. I noen av klassene ble noen av metodene og attributter erstattet. Dette ble gjort ved å lage en metode som heter det samme i barneklassen(den klassen som arver). Dette gjelder også $\_\_init\_\_()$ metoden, derfor vil det oppstå problemer hvis man vil legge inn nye initialattributter, da dette vil erstatte arv-attributtene. et eksempel vil være å gjøre slik

som vist i figur(2)

```
1         class eksempel:
2             def __init__(self):
3                 super().__init__()
4                 self.a = 10
5
```

Figur 2: Lage nye initialattributter og hente de andre fra foreldreklassen

3.2 Simulation

Simulation er klassen som styrer hva som skal kjøres i programmet. Den har følgene metoder,

- *reset_screen*: Fyller skjermen med svart for å overskrive tidligere posisjoner.
- *spawn_boids*: Lager boids hvis man trykker på venstre museknapp.
- *simloop*: Dette er den metoden som kjører kontinuerlig i en while-loop så lenge simulasjonsprogrammet er på.

3.3 MovingObject

MovingObject er en generell funksjon som lager instanser som skal kunne bevege seg, finne distanser til andre bevegelige objekter, unngå å kræsje i vegger og tegne dem. Dette er en klasse man vil arve fra når man skal ha mer spesifikke klasser som bruker alle/mange av metodene og attributtene til MovingObject. MovingObject består av funksjonene,

- *move*: Beveger objektet etter hvor stor hastigheten er.
- *distance*: Returnerer distansen til et annet objekt.
- *crash_wall_check*: Sjekker om objektet er på vei utafor skjermen. Hvis objektet er på vei ut vil hastigheten bli reflektert og demped. Dette skjer ved å gange hastigheten $\boldsymbol{v}$ med en refleksjonskonstant gitt i konfigurasjonefilen.
- *draw*: Tegner objektet ut på skjermen.

Move metoden summer på den tidligere posisjonen og på en enhetsvektor som er skalert av verdi gitt fra konfigurasjonsfilen.

$$\boldsymbol{V} = \frac{\boldsymbol{v}}{||\boldsymbol{v}||}\alpha \tag{1}$$

,der α er en skalar for hva farten skal være slik at,

$$||\boldsymbol{V}|| = \alpha, \forall\, t \tag{2}$$

Metoden *distance* finner distansen til et annet objekt ved ta lengden gitt fra pytagoras,

$$Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

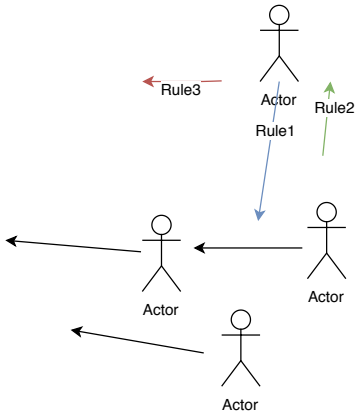
der subindeks {1} angir posisjonen til objektet man vil finne lengden fra og {2} er posisjonen til objektet man vil finne lengden til.

3.4 Boid

Denne klassen inneholder metodene som får objektene til å oppføre seg som en naturlig flokk. Dette blir gjort ved å lage 3 forskjellige regler som kjører kontinuerlig i while-loopen i simulasjons-løkka. Denne klassen arver også fra *MovingObject* derfor vil denne også inneholde alt den også har. Vi har metodene,

- *find_flock*: Putter alle boidsene som tilhører en boid sin flokk i en liste som attributt til en boid. Alle boids vil derfor få en liste med boids(denne tar ikke med boidsen selv).
- *rule1*: Finner gjennomsnitts-posisjonen og lager en vektor dit, denne blir summet på hastighets-vektoren til boidsen.
- *rule2*: Boidsene unngår å kræsje i hverandre ved å summe på en vektor som er i motsatt retning av boidsen som den er på vei å kræsje med.
- *rule3*: Boidsene prøver å matche hastigheten til boidsene som er i sin umiddelbare nærhet.[1]
- *avoid_hawk*: Summer på vektor som er i motsatt retning av haukene/hoikene slik at de prøver å unngå dem.
- *draw*: Tegner dem ut på skjermen.

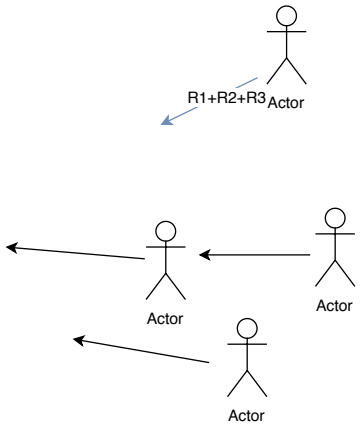
I figur(3) ser man hvordan de tre vektorene kan se ut for en boid.



Figur 3: De forskjellige vektorene fra regel 1,2 og 3

Når disse blir summet opp vil man få en totalvektor som vist i figur(4) under,
Dette gir en fin og glatt bevegelse til flokkene som ser noe naturlig ut.

[1] Disse reglene ble laget ved bruk av informasjonen på `http://www.kfish.org/boids/pseudocode.html`



Figur 4: Summen av vektorene

3.5 Hawk

Denne klassen arver fra boids så disse vil oppføre seg likt som dem. Metodene er,

- *attack*: summer opp en vektor som peker i retningen av den nærmeste boiden til hastigheten til hauken.
- *move*: Lagt til en egen move funksjon fordi hauken har en egen global variabel i konfigurasjonefilen som angir farten uavhengig av boid farten.
- *draw*: Tegner hauken ut på skjermen.

4 Implementasjon

Koden er skrevet i Python versjon 3.[2]
OS: Windows 10
Systemtype: 64-bit OS, x64-basert prosessor
Skjermkort: NVIDIA Geforce 920MX
CPU: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
RAM: 4GB
Pygame [3] Version: 1.9.4
Nympy [4] Version: 1.14.5

[2]https://www.python.org/
[3]https://www.pygame.org/wiki/GettingStarted
[4]http://www.numpy.org/

5 Diskusjon

5.1 Evaluasjon

6 Konklusjon

7 Appendix

8 Referanser