AgentsPy

Release 0.7

Mar 23, 2021

Contents

AgentsPy er et bibliotek til Python, der gør det nemt at arbejde med agent-baseret simulering i undervisning. Det gør det muligt at forstå fænomener fra for eksempel biologi, økonomi, fysik, kemi ved at programmere små simuleringer.

Som eksempel er her en epidemimodel udviklet i AgentsPy. Hvis du selv vil prøve kræfter med modellen, så se epidemi-tutorialen nedenfor.

CHAPTER 1

Første skridt

1.1 Hvad er en agent-baseret model?

Der er mange eksempler på ting i den virkelige verden, som er gavnlige at kunne forudsige eller simulere: det kan for eksempel være huspriser, sygdomsspredning eller selve vejret. For at kunne simulere disse ting kan man bruge en *model*, der kan give en *approksimation* af virkeligheden. Hvis man bruger en model til at simluere et virkeligt fænomen, siges man derfor også at *modellere* fænomenet.

Eksempler på modeller:

- ADAM modellen, er en økonomisk model udviklet af Danmarks Statistik og anvendes af Finansministeriet til samfundsøkonomiske analyser.
- COVID-19 modellerne fra Statens Serum Institut's ekspertgruppe bruges til at afprøve forskellige genåbningsscenarier for pandemien, og estimere sygehusbelastning.

En model kan have mange former. nogle gange er det så simpelt som en matematisk funktion; andre gange er det et helt computerprogram. En af de slags modeller, der ofte findes som et computerprogram, kaldes en **agent-baseret model**.

Se et eksempel på en agent-baseret model af evolution i denne Youtube video:

Med en agent-baseret model forsøger man at modellere et fænomen, som består af et miljø med mange, simple agenter. Agenterne behøver ikke nødvendigvis at repræsentere mennesker, men kan også repræsentere for eksempel elektroner i en ledning eller virksomheder der handler med hinanden i en markedsøkonomi.

En agent-baseret model består typisk af følgende:

- forskellige typer af agenter
- et miljø, hvor agenter færdes
- en beskrivelse af agenternes individuelle adfærd

Som det ses af følgende figur kan agenter både interagere direkte med hinanden, eller indirekte med hinanden ved at interagere med det miljø de færdes i:

Fig. 1: *Illustration fra artiklen "Using AnyLogic and agent-based approach to model consumer market" af Garifullin et al. (2007)*

Agenternes adfærd beskrives som programkode, og det er det vi med *AgentsPy* håber at gøre lidt nemmere og sjovere. *AgentsPy* er et bibliotek til programmeringssproget Python. For at komme i gang, skal du starte med at installere en Python editor ved at følge instruktionerne på en af de følgende sider:

- *Kom godt i gang (Mu-editor)*
- *Kom godt i gang (Thonny)*

1.2 Kom godt i gang (Mu-editor)

1.2.1 Installation af Mu-editoren

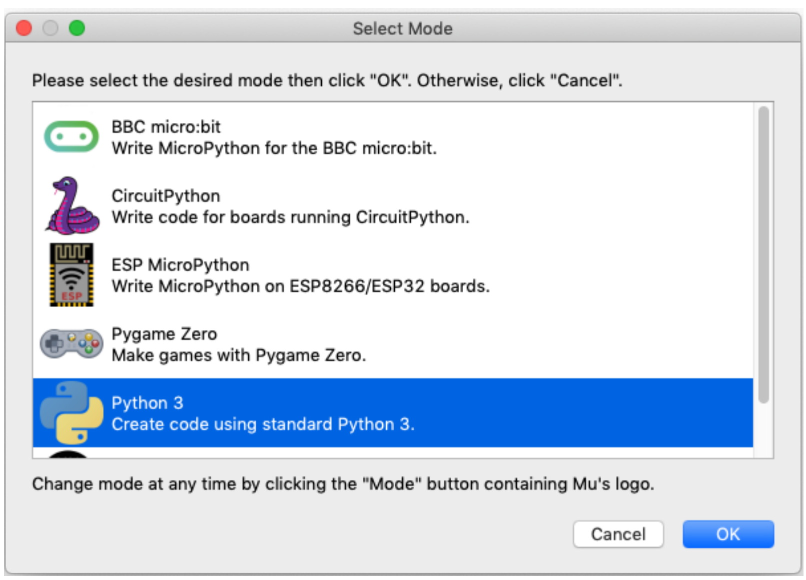
Hent og installér denne beta-udgave af Mu-editoren:

- Hent Mu-editor til Windows (64 bit)
- Hent Mu-editor til Mac OS X
- Hent Mu-editor til Linux

Warning: Nyeste udgave af Mu fra deres hjemmeside er 1 år gammel og understøtter ikke nyeste udgaver af Mac OS X og heller ikke installation af eksterne biblioteker, såsom AgentsPy.

1.2.2 Start Mu

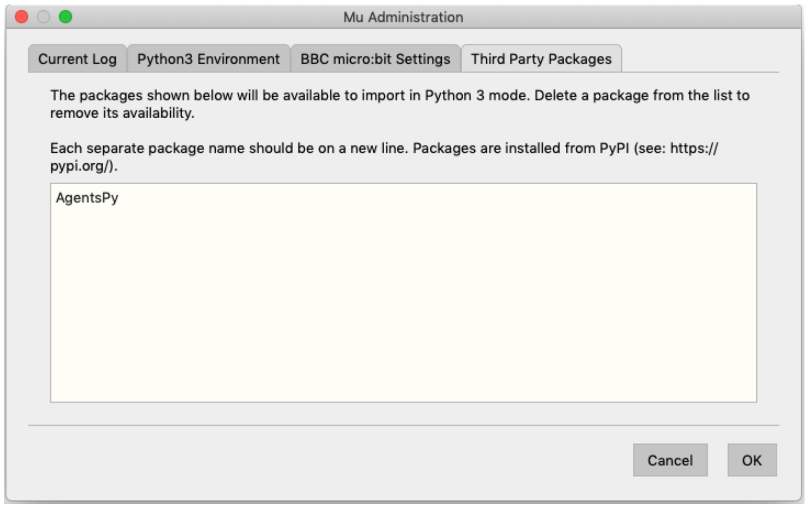
Åbn Mu-editoren. Første gang Mu åbner bliver du bedt om at vælge en *mode* i dialogen **Select Mode**. Her skal du vælge **Python 3** og trykke "OK":



Har du problemer med at åbne Mu på Mac? Læs *fejlsøgningsguiden i bunden af denne side*.

1.2.3 Installer AgentsPy

1. Klik på -ikonet nederst i højre hjørne.
2. Vælg fanen *Third Party Packages*.
3. I tekstfeltet, indtast *agentspy* og klik "OK".



1.2.4 Dit første program med AgentsPy

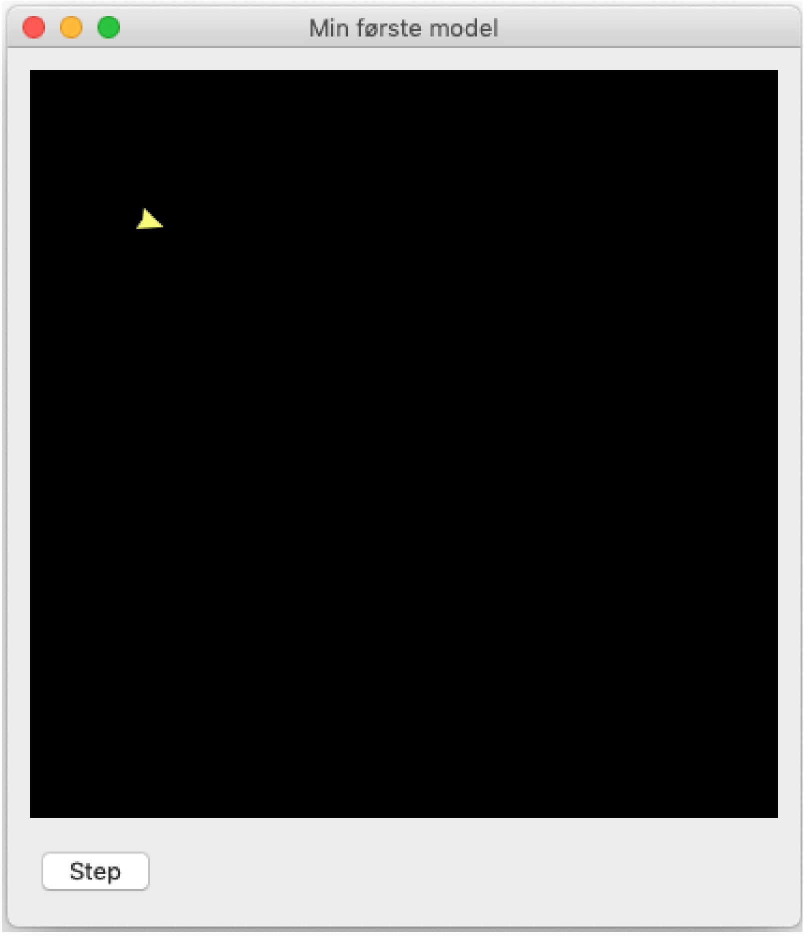
Du er nu klar til at skrive dit første lille agent-baserede program. Du placerer cursoren på linjen efter den hvor der står `# Write your code here :-)`,og skriver følgende:

```
# Importer biblioteket `agents`
from agents import *

# Opret en model og en agent
min_model = Model("Min første model", 50, 50)
min_agent = Agent()

# Tilføj agenten til modellen
min_model.add_agent(min_agent)

# Tilføj en `step`-funktion, og en knap der aktiverer den
def step(model):
  min_agent.forward(10)

min_model.add_button("Step", step)

# Kør modellen
run(min_model)
```

Når du har skrevet ovenstående, kan du prøve programmet ved at trykke på Run .

Du bliver nu bedt om at gemme filen. Gem filen som `agentdemo.py` (**OBS!** Du må IKKE gemme den som `agents.py`)

Du burde nu se følgende vindue:



Prøv at trykke på knappen "Step" et par gange, for at få din agent til at tage et skridt.

Linjerne der starter med # i programmet, bliver forstået som en kommentar til koden, og får ikke betydning for dit program.

1.2.5 Næste skridt

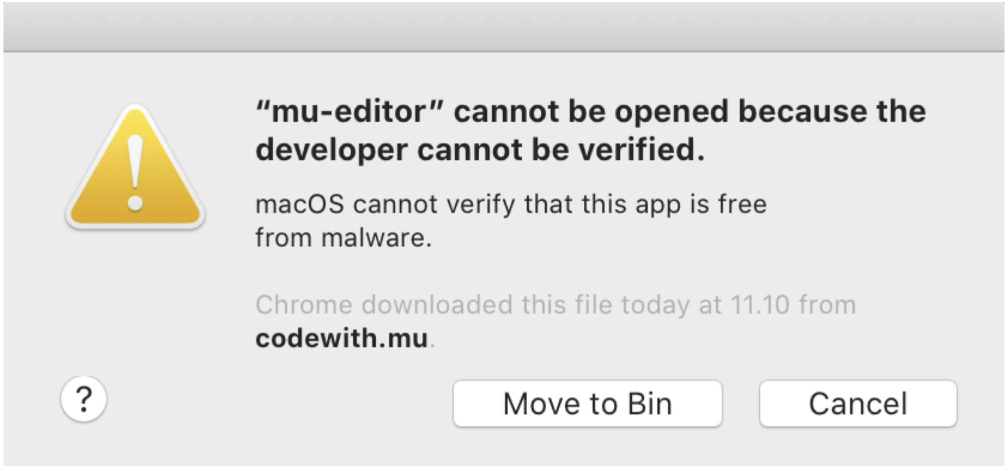
Tillykke du er nu godt igang! Som det næste vil vi anbefale at du følger en af vores tutorials her på siden.

Hvis du vil vide mere om selve Mu-editoren, så har holdet bag Mu-editoren en række tutorials, der kan gøre dig fortrolig med hvordan Mu fungere, de er på engelsk og du finder dem her: https://codewith.mu/en/tutorials/

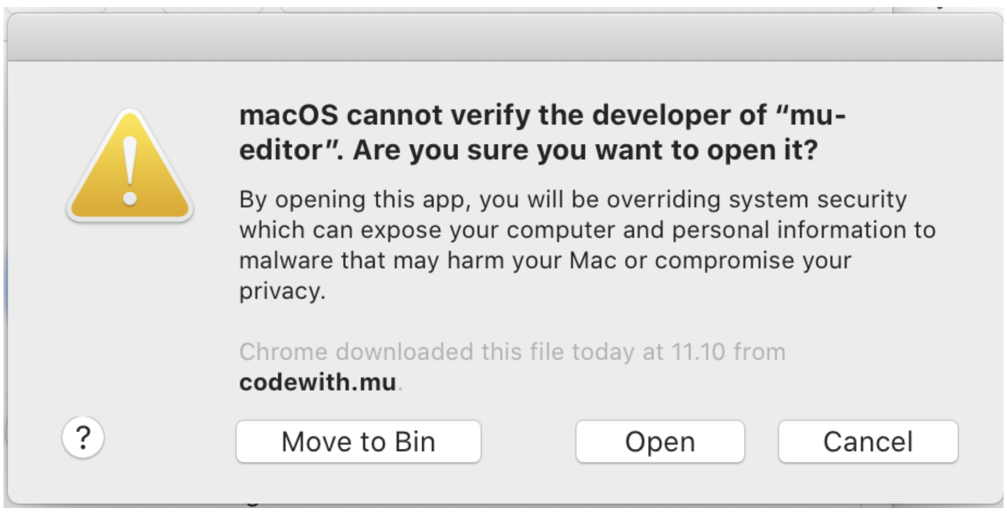
1.2.6 Problemer med at åbne Mu på Mac?

Hvis du er Mac-bruger og ser en besked om, at programmet ikke kan åbnes, fordi det stammer fra en ukendt udvikler eller ikke blev hentet fra App Store, skal du gøre følgende:

- Finde programmet i *Finder*.
- Holde *control* nede og klikke - eller højreklikke, hvis du har mus tilsluttet.
- Der dukker nu en menu frem og øverst kan du vælge *open*.



- Der vil nu dukke et vindue op, hvor du igen vælger *open*
- Fremover vil programmet åbne, som alle andre programmer.



Hvis det stadig ikke virker, så prøv først at genstarte computere, og hvis det så stadig ikke virker, kan du prøve følgende, der slår nogle sikkerhedstjeks fra:

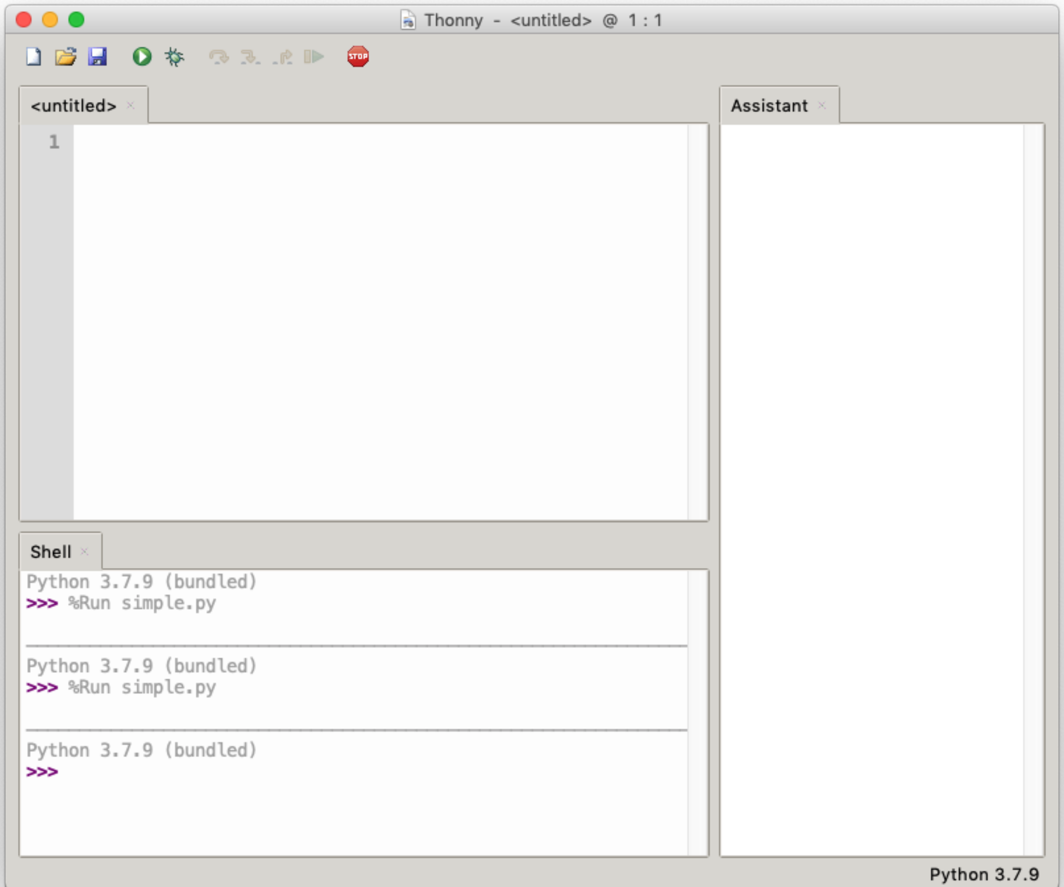
- Åben en Terminal
- Indtast kommandoen `sudo spctl --master-disable`
- Indtast dit password og tryk enter
- For at returnere til de oprindelige indstillinger, kan du køre kommandoen `sudo spctl --master-enable` i en terminal.

1.3 Kom godt i gang (Thonny)

1.3.1 Installation af Thonny

Hent og installer Thonny fra: https://thonny.org/

Åbn Thonny, du burde få et vindue op der ser nogenlunde sådan her ud:



1.3.2 Installer AgentsPy i Thonny

1. Vælg **Tools** -> **Manage Packages**.
2. Skriv `agentspy` i feltet og klik på **Search on PyPI**.
3. Klik på **AgentsPy** og derefter **Install**.

1.3.3 Dit første program med AgentsPy

Du er nu klar til at skrive dit første lille agent-baserede program. Du placerer cursoren på linjen efter den hvor der står `# Write your code here :-)`,og skriver følgende:

```
# Importer biblioteket `agents`
from agents import *

# Opret en model og en agent
```

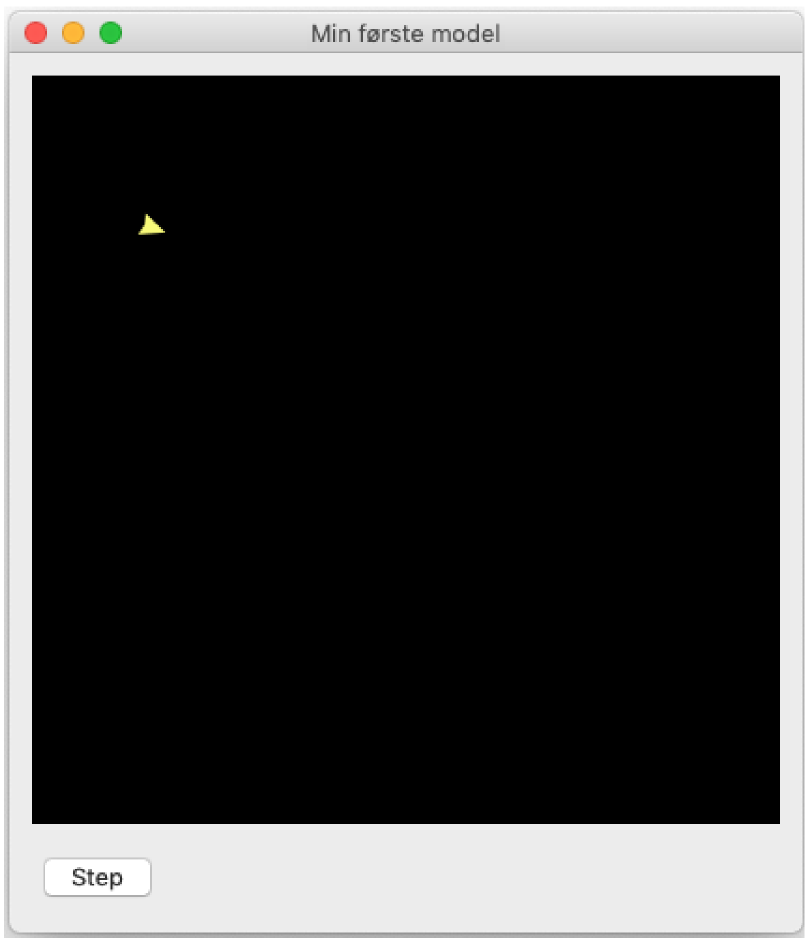
(continues on next page)

(continued from previous page)

```
min_model = Model("Min første model", 50, 50)
min_agent = Agent()

# Tilføj agenten til modellen
min_model.add_agent(min_agent)

# Tilføj en `step`-funktion, og en knap der aktiverer den
def step(model):
  min_agent.forward(10)

min_model.add_button("Step", step)

# Kør modellen
run(min_model)
```

Når du har skrevet ovenstående, kan du prøve programmet ved at trykke på Run .

Du burde nu se følgende vindue:



Prøv at trykke på knappen "Step" for at få din agent til at tage et skridt.

1.3.4 Næste skridt

Tillykke du er nu godt igang! Som det næste vil vi anbefale at du følger en af vores tutorials her på siden.

Hvis du vil vide mere om selve Mu-editoren, så har holdet bag Mu-editoren en række tutorials, der kan gøre dig fortrolig med hvordan Mu fungere, de er på engelsk og du finder dem her: https://codewith.mu/en/tutorials/

1.4 Introduktion til Python

Hvis du ikke endnu har installeret en editor, så brug en af følgende guides:

- *Kom godt i gang (Mu-editor)*
- *Kom godt i gang (Thonny)*

1.4.1 Turtle-biblioteket

Som indledning til at bruge det agent-baserede bibliotek `AgentsPy` vil vi bruge et andet bibliotek, kaldet `turtle`. Et bibliotek er en samling af eksterne funktioner, som man kan bruge i sit eget program.

Start med at åbne din editor, lav en ny fil, og gem den som `turtle_test.py`.

Lad os nu prøve at kode med `turtle` biblioteket. For at bruge et bibliotek, skal man først *importere* det. Tilføj følgende linje kode til din fil:

```
from turtle import *
```

Stjernen `*` indikerer, at vi gerne vil importere alle funktioner fra biblioteket. Ovenstående linje kode gør ikke noget af sig selv, men efter importeringen kan du nu fremover i din fil bruge funktioner fra `turtle` biblioteket.

Lad os nu lave en "turtle". En turtle er en lille agent (markeret med en pil), som kan flyttes rundt på en skærm ved at kalde nogle bestemte funktioner. Lav en turtle ved nederst i filen at skrive:

```
t = Turtle()
```

Når du kører din fil, burde der komme et vindue frem med en hvid baggrund og en sort pil i midten. Den sorte pil er dit "turtle-objekt", som kan refereres med variablen `t`.

Luk vinduet, og tilføj denne linje kode til filen:

```
t.forward(100)
```

Kører du filen, burde gerne se din turtle rykke sig lidt fremad. Giv den lidt flere instrukser:

```
t.left(90)
t.color("red")
t.forward(200)
```

Det her er bare nogle af de funktioner, man kan bruge på sin "turtle" (agenterne fra `AgentsPy` har nogle lignende funktioner).

Opgave 1

Brug `t.forward()` og `t.left()` til at få turtle-objektet til at tegne en firkant.

Hint: hver funktion skal kaldes flere gange.

1.4.2 Egne funktioner

Indtil videre har vi kun brugt eksisterende funktioner fra biblioteket, men det er også muligt at lave sine egne funktioner. Funktioner definerer en sekvens af kode, som man kan køre gentagne gange ved at "kalde" funktionen.

Vi laver nu vores egen funktion, kaldet `draw_square()`, som tegner en firkant. Slet din eksisterende kode, *undtagen* den øverste linje, hvor du importerer `turtle` biblioteket. Begynd så med at tilføje denne linje, der erklærer funktionen:

```
def draw_square(turtle):
```

I denne funktion er `turtle` et *argument*, som man kan give med til funktionen, når man kalder den. For at sammenligne: ved funktionskaldet `t.color("red")` er det `"red"`, som er argumentet. I dette tilfælde er argumentet den turtle, som vi bruger til at tegne firkanten.

Tilføj nu de følgende linjer kode lige under funktionserklæringen:

```
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
```

Koden får turtle-objektet til at dreje sig 90 grader og gå 100 skridt frem, fire gange.

Det er vigtigt, at kode, som er en del af funktionen, rykkes ind ved at sætte mellemrum foran, sådan at det står længere til højre end selve funktionserklæringen. Hele funktionen skal altså se således ud:

```
def draw_square(turtle):
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
```

Prøv nu at lave et turtle-objekt, gemt i variablen `t`, og *kald* så funktionen ved at skrive:

```
draw_square(t)
```

Bemærk, at en funktion kan kun kaldes, efter at den er blevet erklæret, så ovenstående linje kode skal stå *under* funktionen selv.

1.4.3 Loops

Ved et nærmere kig på `draw_square()` funktionen er den ikke særlig "smart" skrevet. Koden, der tegner en linje, er ens for hver af de fire linjer. Vi kan gøre det smartere ved at lave et *loop*, der i stedet kører koden for hver linje 4 gange.

Slet indholdet af `draw_square()`, og skriv i stedet følgende linje (husk at rykke linjen ind til højre, så den stadig er "inde" i `draw_square()`):

```
for i in range(4):
```

Dette "for-loop" gentages 4 gange. Ligesom at vi kan have kode inde i funktioner, kan vi også have kode inde i loops. Lav efter ovenstående linje et nyt indryk, og skriv:

```
turtle.left(90)
turtle.forward(100)
```

Nu burde hele funktionen se sådan ud:

```
def draw_square(t):
    for i in range(4):
        turtle.left(90)
        turtle.forward(100)
```

Kører du koden igen, burde du gerne få samme resultat.

Opgave 2

Prøv at lave en funktion, `draw_circle()`, der ligesom `draw_square()` tager et turtle-objekt, men i stedet tegner en cirkel.

Hint: du behøver kun at ændre på nogle af tallene i `draw_square()`.

1.4.4 If-sætninger

Det er også muligt at have kode, som kun bliver kørt, hvis nogle bestemte kriterier er opfyldt. Lad os, for at demonstrere, prøve at lave en funktion, der tegner et "S". Start med at erklære en funktion `draw_S()`, der tager en turtle `t` som argument. Lav så et for-loop i den, der kører 360 gange. Tilføj inde i loopet følgende kode:

```
t.forward(1)
if i < 180:
    t.left(1)
else:
    t.right(1)
```

Når vi skriver `for i in range(360)`, kommer variablen `i` til at antage værdierne fra 0 til 359. Vores "*if-sætning*" checker, om `i` er større eller mindre end 180. Hvis `i` er mindre, drejer vores turtle til venstre, ellers drejer den til højre.

Prøv at kalde funktionen og se, om din turtle tegner noget, der ligner et "S".

Opgave 3

Vi udvider nu `draw_square()` funktionen, sådan at den kan lave en firkant af en bestemt størrelse, som brugeren giver, *hvis* der gives en gyldig størrelse.

Vi gør først sådan, at funktionen tager *to* argumenter i stedet for kun ét. Ændr i funktionsdefinitionen, sådan at den tager et ekstra argument, `size`:

```
def draw_square(t, size):
```

Gør nu sådan, at turtle-objektet bevæger sig en længde på `size` frem i hvert loop, i stedet for 100.

Hint: du behøver kun at ændre i kaldet til `t.forward()`.

Det giver ikke mening at tegne en firkant, hvor størrelsen på siderne er negative. Brug derfor en if-sætning til at sikre, at *hele* for-loopet kun køres, hvis `size` er større end 0.

Hint: husk at sikre, at din kode har din rigtige indrykning.

Opgave 4

Vi laver funktionen `walk_random()`, der får den pågældende turtle til at gå tilfældigt rundt på skærmen.

Tilføj først denne linje til dine "imports", altså lige under den linje, hvor du importerer `turtle`-biblioteket:

```
from random import randint
```

`randint(a,b)` giver et tilfældigt tal mellem `a` og `b`.

Lav nu funktionserklæringen til `walk_random()`:

```
def walk_random(t):
```

I selve funktionen, lav et for-loop, der kører i 500 iterationer. Inde i for-loopet, få `t` til at bevæge sig fremad, og lav så en if-sætning, der checker, om et tilfældigt tal er 1 eller 0. Hvis tallet er 1, skal `t` dreje 10 grader til venstre, ellers 10 grader til højre.

1.5 Tutorial: Epidemi-model

Tillykke! Du er, i midten af en verdensomspændende pandemi, netop blevet ansat som den nye direktør for sundhedsstyrelsen. Regeringen har givet dig din første opgave: forudsig, hvordan sygdommen spreder sig, og kom med forslag, der kan mindske smittespredningen.

Du sidder længe og grubler over, hvordan du skal forudsige spredningen, da en af dine kollegaer pludselig kommer med et godt forslag. De foreslår, at du programmerer en *agent-baseret model*, som kan simulere smittespredningen. På den måde kan du så bruge modellen til at forudse, hvad der kommer til at ske i den virkelige verden. Du tænker, at dette lyder som en fantastisk ide, og går straks i gang med at kode en simpel model.

Denne tutorial er delt i fire dele. I de første to dele opbygges grundmodellen. I de sidste to introduceres forskellige variationer over den indledende model.

1.5.1 Del 1: Agenter i Python

Den første agent

Før vi begynder at lave agenter, der kan simulere smittespredning, skal vi først have en *model*, vi kan have dem i. Begynd med at lave en fil, kaldet `epidemic.py`, og giv den følgende indhold:

```
from agents import *

epidemic_model = Model("Epidemi-model", 50, 50)

run(epidemic_model)
```

Linje 1 gør sådan, at alle funktionaliteterne i biblioteket `AgentsPy` kan bruges i filen. Det er det bibliotek, der giver adgang til alle de nødvendige funktioner.

Linje 3 laver en model med 50x50 felter (*tiles*), og navnet *Epidemi-model*.

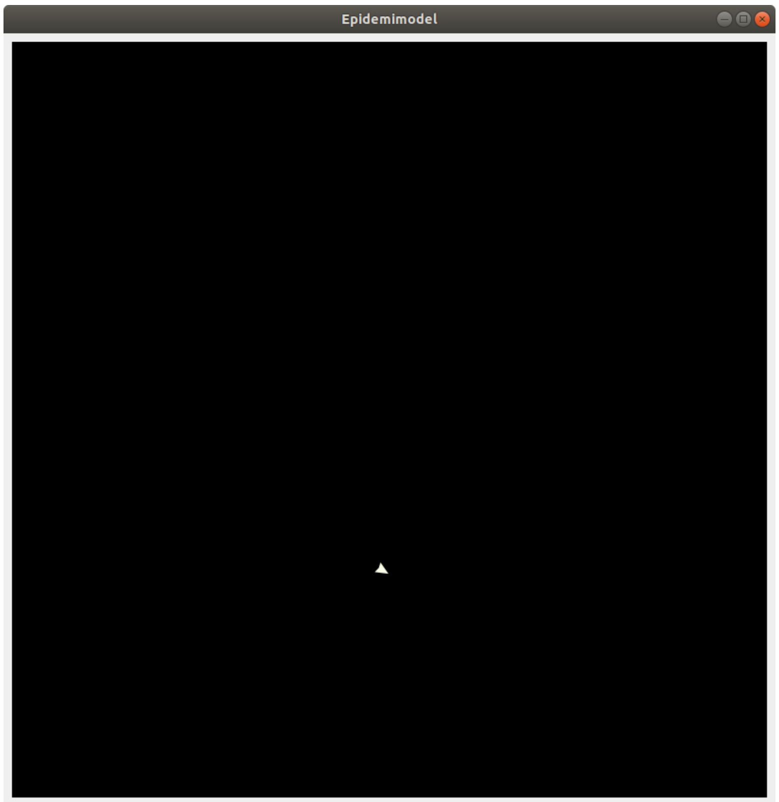
Linje 5 starter modellen.

Prøv at køre programmet, og se, hvad der sker. Der burde vises et vindue af en sort firkant. Dette er en tom model.

Tilføj nu, på linje 4, følgende kode:

```
min_agent = Agent()
epidemic_model.add_agent(min_agent)
```

Disse to linjer laver en agent ved at bruge `Agent()`, og tilføjer den så til modellen ved at bruge `add_agent()`. Starter man modellen igen, burde der vises en enkelt lille trekant inde i modellen - dette er agenten.



Knapper

For at gøre det nemmere at styre vores model undervejs, vil vi gerne tilføje nogle knapper til vinduet, som man kan klikke på for blandt andet at starte og stoppe simulationen.

Lad os først tilføje en *setup* knap, som genstarter modellen. Indtil videre skal den bare slette alle eksisterende agenter, og lave en ny.

Slet først de sidste to linjer, du tilføjede ovenfor (altså dem, der laver en agent og tilføjer den til modellen). Tilføj så denne funktion, lige efter, at du har importeret `agents`:

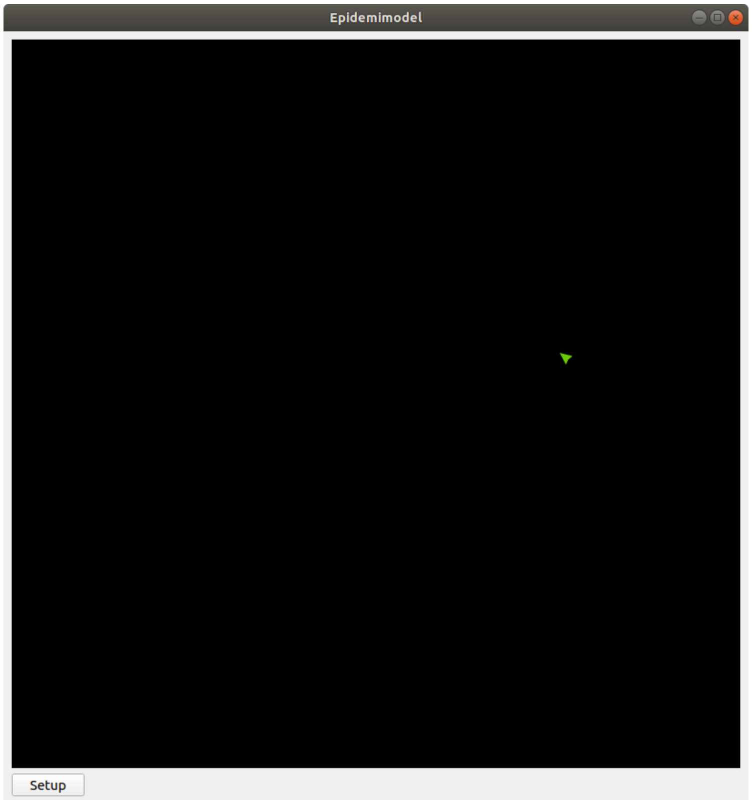
```
def model_setup(model):
    model.reset()
    model.add_agent(Agent())
```

Funktionen her sletter alle agenter med `model.reset()` og tilføjer en ny med `model.add_agent()`. Det kan virke lidt ligegyldigt nu, men det vil blive brugbart senere.

Tilføj så, efter du har lavet `epidemic_model`, følgende linje:

```
epidemic_model.add_button("Setup", model_setup)
```

Linjen tilføjer en knap til vinduet som, når den klikkes på, kører `model_setup`-funktionen.



Flere agenter

Lad os tilføje lidt flere agenter. Ændr `model_setup` funktionen, sådan at den siger følgende:

```
def model_setup(model):
    model.reset()
    for agent in range(100):
        model.add_agent(Agent())
```

Nu laver vi 100 agenter og tilføjer dem til modellen.

Lige nu laver agenterne ikke særlig meget. Lad os gøre det muligt for agenterne at gå rundt omkring. Tilføj denne `model_step` funktion under `model_setup` funktionen:

```
def model_step(model):
    for agent in model.agents:
        agent.direction += randint(-10,10)
        agent.forward()
```

Vi gennemgår funktionen:

- For hver agent i modellen:
  - Juster dens retning med en tilfældig vinkel mellem -10 og 10.
  - Ryk den et skridt fremad i den retning, den peger.

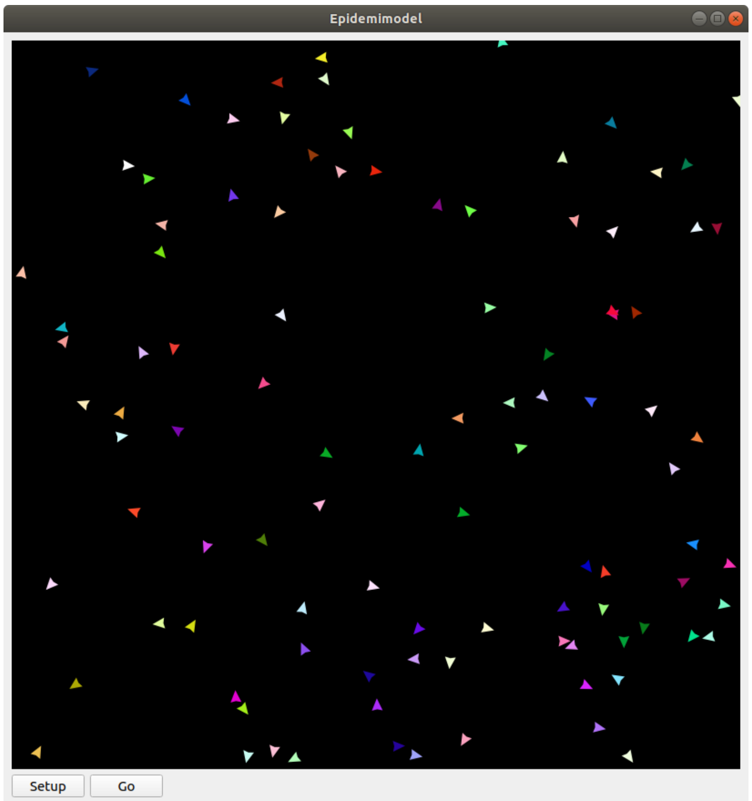
`randint(a,b)` er en funktion, det vælger et tilfældigt tal mellem `a` og `b`. For at bruge den, skal du lige importere den (gør dette i toppen af filen, sammen med at du importerer `agents`):

```
from random import randint
```

Slut af med at tilføje denne linje efter at du tilføjer *setup*-knappen:

```
epidemic_model.add_button("Go", model_step, toggle=True)
```

Dette laver en knap, som man kan slå til og fra. Når den er slået til, kører den `model_step`-funktionen konstant, hvilket får agenterne til at bevæge sig rundt.



Samlet kode

Her er den samlede kode du gerne skulle have nu:

```
from agents import *
from random import randint

# Opret model
epidemic_model = Model("Epidemi-model", 50, 50)

# Reset model
def model_setup(model):
    model.reset()
    for agent in range(100):
        model.add_agent(Agent())
```

(continues on next page)

(continued from previous page)

```
# Tag et skridt i modellen
def model_step(model):
    for agent in model.agents:
        agent.direction += randint(-10,10)
        agent.forward()

# Tilføj knapper til reset og go
epidemic_model.add_button("Setup", model_setup)
epidemic_model.add_button("Go", model_step, toggle=True)

# Kør modellen
run(epidemic_model)
```

1.5.2 Del 2: En model over smittespredning

Du har nu din model, og dine agenter - men hvordan skal du simulere sygdommen? Du grubler meget længe, indtil at en anden kollega fortæller dig om SIR-modellen [#]_ : en matematisk model, som bruges til at modellere sygdomsspredning.

Modellen har tre kategorier, som den opdeler folk i:

- Susceptible: Folk i denne gruppe er modtagelige, og kan blive smittet, hvis de kommer i kontakt med en, der bærer sygdommen.
- Infectious: Folk i denne gruppe er blevet syge, og kan smitte folk, der er modtagelige.
- Recovered: Folk i denne gruppe har haft sygdommen og er blevet raske og immune, og kan derfor ikke længere hverken smitte eller blive smittet.

En person kan altså kun være i én kategori ad gangen, og deres tilstand vil have mønsteret:

Susceptible → Infectious → Recovered

Du tænker, at dette er lige den model, du har brug for, og går straks i gang med at kode.

Fra agent til person

Lige nu er vores agenter "bare" agenter. Vi vil gerne gøre dem lidt mere avancerede, sådan at de blandt andet kan selv kan holde styr på, hvilken kategori af SIR-modellen, de er i.

Tilføj, over din `model_setup`-funktion (men under dine imports), følgende kode:

```
class Person(Agent):
    def setup(self,model):
        self.category = 0

    def step(self,model):
        self.direction += randint(-10,10)
        self.forward()
```

Ovenstående kode definerer en *klasse*, som har noget opførsel beskrevet i sine egne funktioner `Person.setup` og `Person.step`.

Ændr så `model_setup`-funktionen til:

```
def model_setup(model):
    model.reset()
    for person in range(100):
        model.add_agent(Person())
```

Nu tilføjer vi altså personer i stedet for "bare" normale agenter.

Bemærk, at indholdet i `Person.step` lidt ligner det, der står i `model_step`-funktionen i forvejen. Faktisk kan vi nu også ændre i `model_step`-funktionen, sådan at der i stedet står:

```
def model_step(model):
    for person in model.agents:
        person.step(model)
```

Prøv nu at køre modellen igen. Hvis du har gjort det rigtigt, burde den ikke se anderledes ud end før.

Kategorier

For ikke at skulle skrive navnene på kategorierne hele tiden, bruger vi i stedet tal, sådan at

Kategori	#
Susceptible	0
Infectious	1
Recovered	2

Tilføj nu en `infect`-funktion til `Person`, som har følgende udseende:

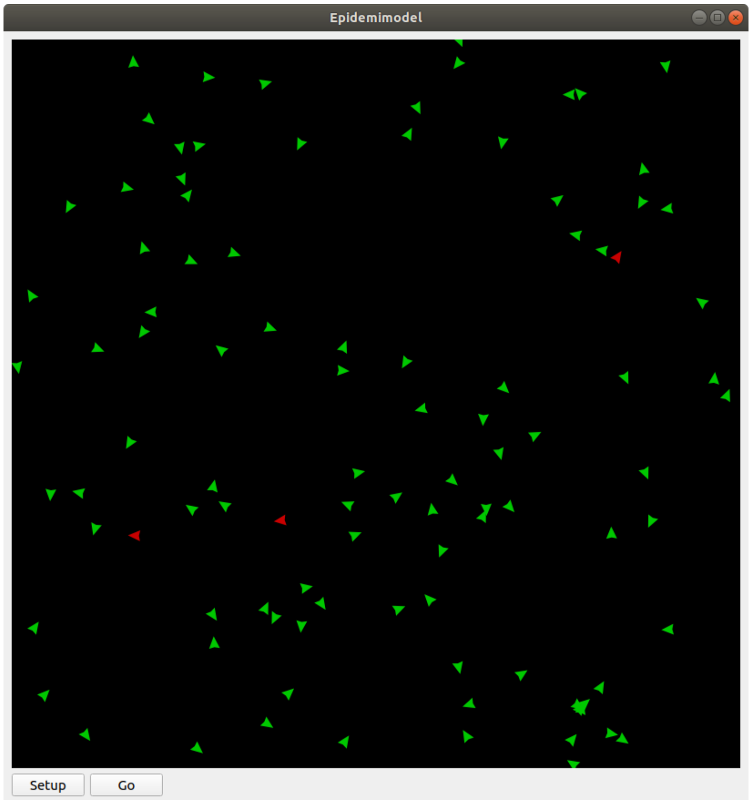
```
def infect(self, model):
    self.color = (200, 0, 0)
    self.category = 1
```

Funktionen giver agenten en rød farve, og sætter den i kategori 1.

Omskriv så `Person.setup` til følgende:

```
def setup(self,model):
    self.category = 0
    self.color = (0, 200, 0)
    if randint(1,50) == 1:
        self.infect(model)
```

Vi gør her sådan, at de fleste agenter starter med at være raske og have en grøn farve, men en lille del (omkring 2%) starter med at være syge og have en rød farve.

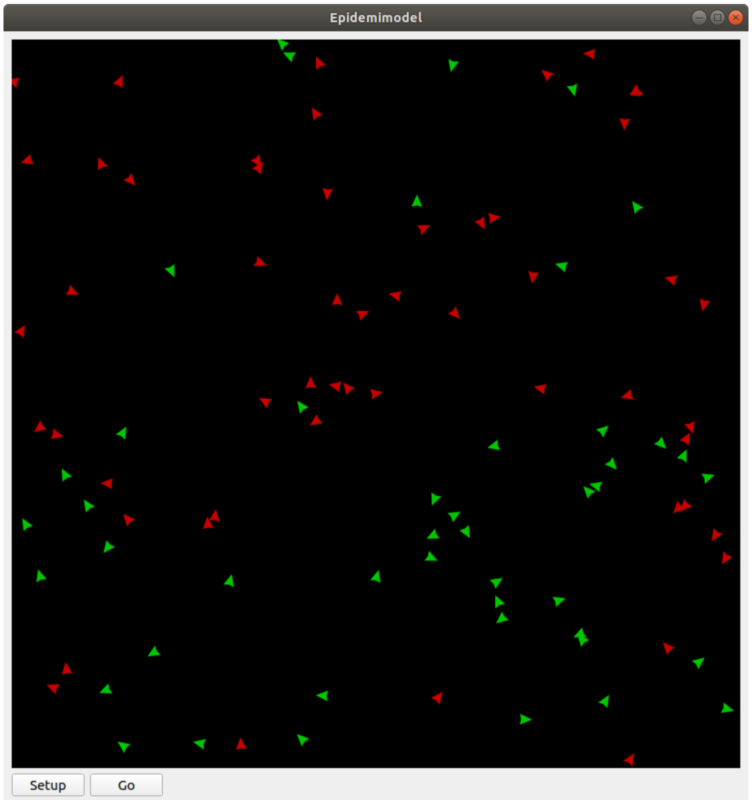


Smittespredning

Ideen med modellen er, at de syge agenter skal smitte de raske agenter. Vi gør det på den måde, at en syg agent smitter alle raske agenter, som er indenfor en bestemt afstand af den. Tilføj følgende kode i bunden af `Person.step`-funktionen:

```
if self.category == 1:
    for agent in self.agents_nearby(12):
        if agent.category == 0:
            agent.infect(model)
```

Koden siger, at hvis agenten er i kategori 1 (altså syg), så smitter den alle agenter indenfor en radius af 12 (agentens egen radius er på 4).



Immunitet

Lige nu kan vores model vise 2 af de 3 kategorier, altså "susceptible" og "infectious". Som det sidste led i modellen, skal agenter i "infectious" kategorien flyttes til "recovered" kategorien, når der er gået et stykke tid.

Tilføj først først denne funktion `turn_immune` til `Person`:

```
def turn_immune(self, model):
    self.color = (0,0,200)
    self.category = 2
```

Denne minder om `Person.infect`, men i stedet for at personen bliver rød og inficeret, bliver den blå og opnår immunitet.

Tilføj så denne linje til `Person.infect`:

```
self.infection_level = 600
```

Idéen med `infection_level`-variablen er, at den langsomt tæller ned, og, når den rammer 0, bliver den inficerede agent immun. Det gør vi ved at tilføje disse tre linjer i bunden af `if`-sætningen i `Person.step`:

```
self.infection_level -= 1
if self.infection_level == 0:
    self.turn_immune(model)
```

`if`-sætningen burde til slut gerne se således ud:

```
if self.category == 1:
    for agent in self.agents_nearby(12):
```

(continues on next page)

(continued from previous page)

```
        if agent.category == 0:
            agent.infect(model)
    self.infection_level -= 1
    if self.infection_level == 0:
        self.turn_immune(model)
```

Når du kører programmet, burde du nu have en færdig implementation af SIR-modellen.

Grafer

Til slut vil vi gerne se, om vores model forløber på samme måde som SIR-modellen. Det gør vi ved at indsætte en graf, som viser fordelingen af agenter over tid.

Ideen med grafen kommer til at være, at vi optæller antallet af agenter i hver kategori, og så får grafen til at vise tre linjer, som viser antallene i hver kategori som funktion af tid.

Begynd først med at indsætte disse tre linjer i `model_setup`-funktionen, lige efter du har kaldt `model.reset()`:

```
model.Susceptible = 0
model.Infectious = 0
model.Recovered = 0
```

Vi får agenterne selv til at tildele sig de forskellige kategorier, så vi lader alle tre starte med at være 0.

Tilføj øverst i `Person.setup`:

```
model.Susceptible += 1
```

Tilføj øverst i `Person.infect`:

```
model.Susceptible -= 1
model.Infectious += 1
```

Tilføj øverst i `Person.turn_immune`:

```
model.Infectious -= 1
model.Recovered += 1
```

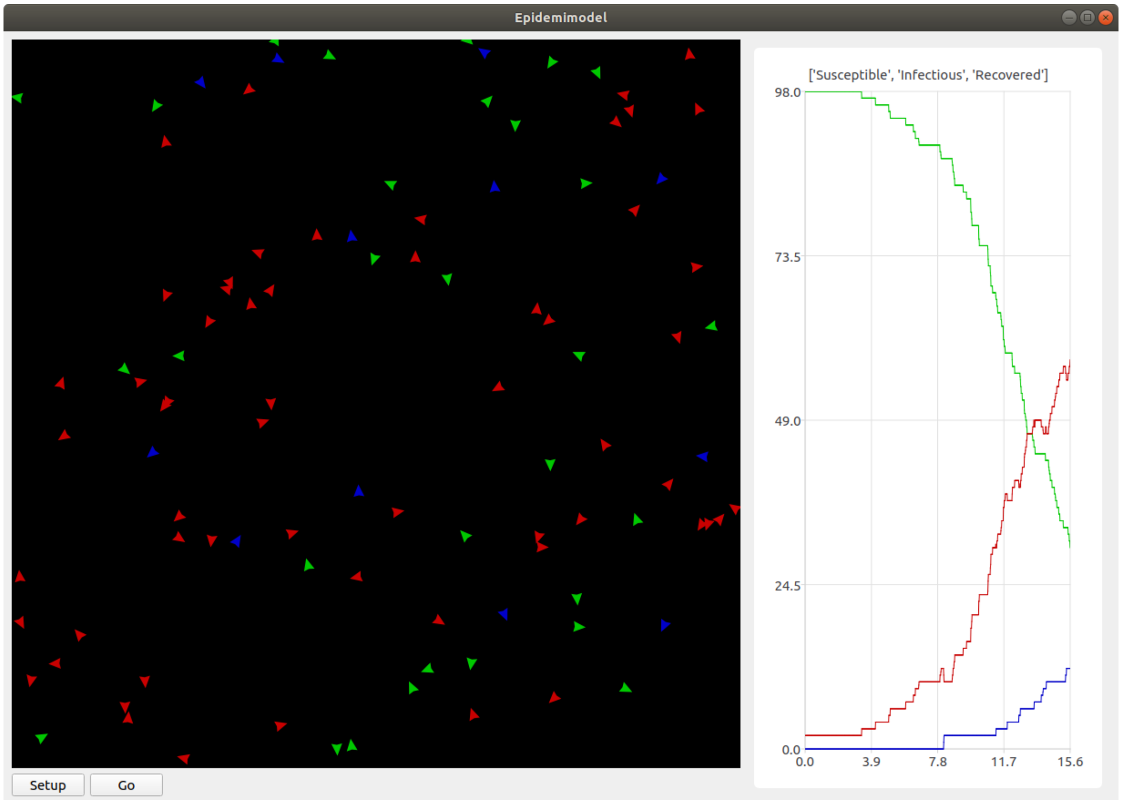
Nu har vi styr på dataen til vores model. Programmet skal dog lige vide, at det skal opdatere grafen, imens *Go*-knappen holdes inde. Tilføj denne linje nederst i `model_step`-funktionen:

```
model.update_plots()
```

Det eneste, vi mangler nu, er at tilføje graferne. Indsæt disse linjer, lige efter der hvor du tilføjer knapperne til modellen:

```
epidemic_model.line_chart(["Susceptible","Infectious","Recovered"],[(0, 200, 0),(200,␣
↪0, 0),(0, 0, 200)])
epidemic_model.bar_chart(["Susceptible", "Infectious", "Recovered"], (200, 200, 200))
```

Prøv at køre modellen, indtil der ikke er flere inficerede agenter tilbage, og sammenlign så den graf du får med den, der er på Wikipedia-siden for SIR-modellen.



Forskelle mellem SIR-modellen og den agent-baserede model

Du har nu udviklet en agent-baseret model, der approksimerer SIR-modellen. Men hvad er fordelene og ulemperne egentlig ved at bruge de to forskellige modeller?

SIR-modellen

- *Fordele*: Da SIR-modellen er baseret på et sæt matematiske formler, er det nemmere at beregne direkte på modellens resultater, samt at kombinere og sammenligne den med andre matematiske modeller. Derudover er modellen også konsistent: hvis man giver det samme input flere gange til den samme model, vil man altid få det samme output.
- *Ulemper*: SIR-modellen er en meget generaliserende model, da den antager, at alle individer opfører sig ens, for eksempel ved at alle har samme infektionsrate og sygdomsforløb. Man kan approksimere forskellige opførsler ved at justere på parametrene, men det er svært at sige, hvor meget det hænger sammen med virkeligheden.

Agent-baseret model

- *Fordele*: Med den agent-baserede model er det nemmere at modellere både forskellige typer adfærd og karakteristika for både personer og virus, for eksempel superspredere, virusmutationer, social afstand, og så videre. Selvom modellen aldrig kan være helt præcis, kan den stadig give god indsigt i, hvilken indflydelse disse faktorer har på epidemien, og hvordan de interagerer med hinanden.
- *Ulemper*: Den agent-baserede model har en stor tilfældighedsfaktor i sig. Agenterne starter tilfældige steder, bevæger sig tilfældigt, og smittes tilfældigt. Derfor er det nødvendigt at køre modellen mange gange for at fastlægge endelige resultater, og selv da kan man ikke garantere det. Derudover er det også svært at kombinere den agent-baserede model med andre eksisterende matematiske modeller.

Opgave 1

Tilføj en tilfældighed, så smitte ikke spredes med 100% sandsynlighed, men fx kun 20% sandsynlighed, når to agenter mødes.

Hint: Brug `randint()`-funktionen, som vi også har brugt tidligere.

Opgave 2

Overvej hvordan vi kan lave en type agent "Superspreder", der enten:

- Bevæger sig hurtigere (flere kontakter)
- Smitter mere end andre agenter (høj smitterate)

Samlet kode

Her er den samlede kode du gerne skulle have nu:

```
from agents import Model, Agent, run
from random import randint


class Person(Agent):
    def setup(self, model):
        model.Susceptible += 1
        self.category = 0
        self.color = (0, 200, 0)
        if randint(1, 50) == 1:
          self.infect(model)

    def step(self, model):
        self.direction += randint(-10, 10)
        self.forward()
        if self.category == 1:
            for agent in self.agents_nearby(12):
                if agent.category == 0:
                    agent.infect(model)
            self.infection_level -= 1
            if self.infection_level == 0:
                self.turn_immune(model)

    def infect(self, model):
        model.Susceptible -= 1
        model.Infectious += 1
        self.color = (200, 0, 0)
        self.category = 1
        self.infection_level = 600

    def turn_immune(self, model):
        model.Infectious -= 1
        model.Recovered += 1
        self.color = (0, 0, 200)
        self.category = 2


def model_setup(model):
```

(continues on next page)

(continued from previous page)

```
    model.reset()
    model.Susceptible = 0
    model.Infectious = 0
    model.Recovered = 0
    for person in range(100):
        model.add_agent(Person())


def model_step(model):
    for person in model.agents:
        person.step(model)
    model.update_plots()


epidemic_model = Model("Epidemimodel", 100, 100)

epidemic_model.add_button("Setup", model_setup)
epidemic_model.add_button("Go", model_step, toggle=True)
epidemic_model.line_chart(
    ["Susceptible", "Infectious", "Recovered"], [(0, 200, 0), (200, 0, 0), (0, 0,␣
↪200)]
)
epidemic_model.bar_chart(["Susceptible", "Infectious", "Recovered"], (200, 200, 200))

run(epidemic_model)
```

1.5.3 Del 3: Mindskning af smitte

Succes! Regeringen er godt tilfreds med din model, der viser spredningen af smitte, og efterfølgende immunitet, over tid. Nu har de givet dig en ny opgave: kom på tiltag til at begrænse smitten, og simulér dem så i modellen, for at se, om de faktisk virker. Heldigvis har dine kollegaer en masse idéer til, hvordan man kan mindske smittespredning.

Hold afstand

Forslag: Agenter prøver på at undvige andre syge agenter.

Vi vil gøre sådan, at alle agenter, der ser en syg agent indenfor en vis afstand, vender sig om og går i den modsatte retning.

Erstat denne linje i `Person.step`:

```
self.direction += randint(-10,10)
```

med disse:

```
avg_direction = 0
nearby_agents = 0
for agent in self.agents_nearby(20):
    if agent.category == 1:
        avg_direction += self.direction_to(agent.x,agent.y)
        nearby_agents += 1
if nearby_agents > 0:
    self.direction = (avg_direction / nearby_agents) + 180
else:
    self.direction += randint(-10,10)
```

Det virker af meget, men ovenstående kode er faktisk ikke så indviklet.

Vi laver først to variabler, `avg_direction` og `nearby_agents`, hvor den første kommer til at indeholde den gennemsnitlige retning til alle de smittede agenter, og `nearby_agents` indeholder antallet af smittede agenter tæt på.

Derefter undersøger vi agenter i nærheden, også dem, som er udenfor smitteradius. Hvis der er en smittet agent, lægger vi retningen til agenten til `avg_direction`, og 1 til `nearby_agents`.

Når alle agenterne er blevet undersøgt, skal vi ændre retning. Hvis der ingen smittede agenter er tæt på, justerer vi bare, som normalt, den nuværende retning med op til 10 grader. Hvis der *er* smittede agenter, finder vi den gennemsnitlige retning med

```
epidemic_model.add_checkbox("enable_groups")
```

Nu kan vi gå i gang med faktisk at lave gruppefunktionaliteten. Tilføj, nederst i `Person.setup`, denne linje:

```
if model.enable_groups:
    self.group = randint(1,5)
```

Dette tildeler agenten til en tilfældig gruppe, identificeret med et ID mellem 1 og 5.

For at vi kan se forskel på de forskellige grupper, tegner vi en cirkel udenom agenterne, hvor farven på cirklen afhænger af deres gruppe. Agenter i samme gruppe har således samme farvecirkel. Tilføj disse linjer kode til `if`-sætningen:

```
self.group_indicator = model.add_ellipse(self.x-10,self.y-10,20,20,(0,0,0))
if self.group == 1:
    self.group_indicator.color = (200,200,0)
elif self.group == 2:
    self.group_indicator.color = (0,200,200)
elif self.group == 3:
    self.group_indicator.color = (200,0,200)
elif self.group == 4:
    self.group_indicator.color = (100,100,100)
elif self.group == 5:
    self.group_indicator.color = (250,150,0)
```

Dette gemmer agentens farvecirkel i variablen `group_indicator`, og giver den en farve afhængigt af `group`-id'et.

Ændr så linje i `Person.step`:

```
if agent.category == 1:
```

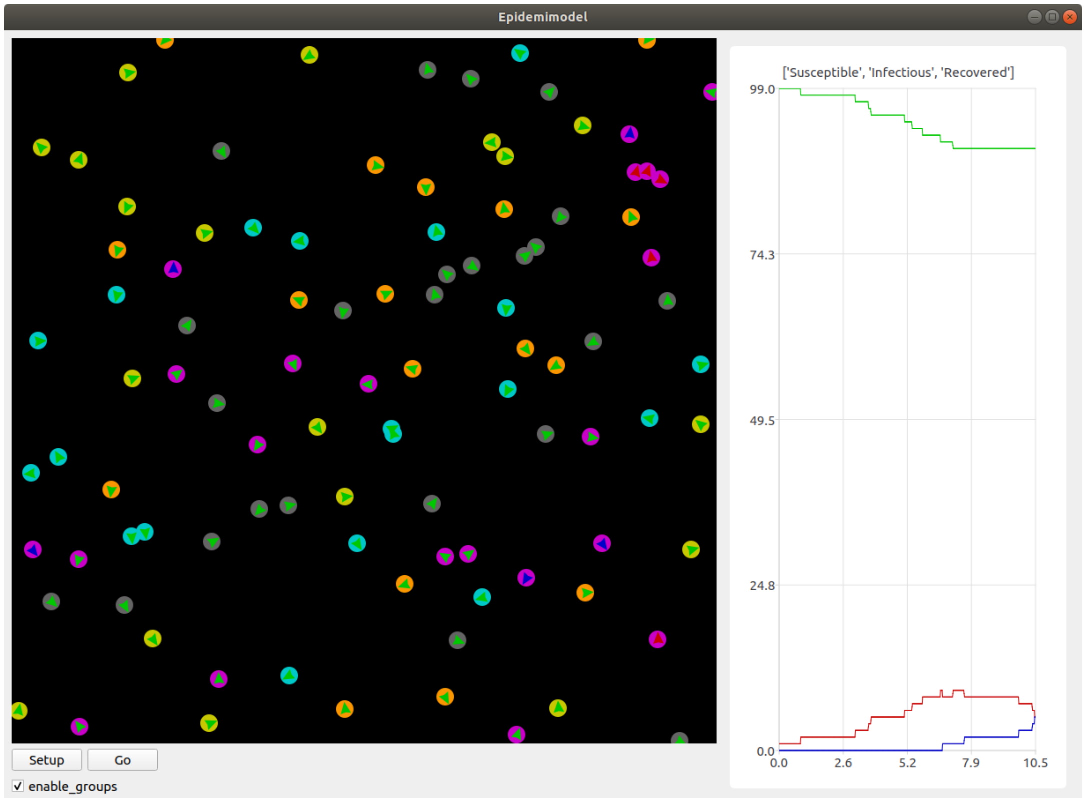
til denne:

```
if model.enable_groups and agent.group != self.group:
```

Det får agenten til at undgå alle, der ikke er i dens egen gruppe, fremfor dem der er smittede. Tilføj til sidst, nederst i `Person.step`:

```
if model.enable_groups:
    self.group_indicator.x = self.x-10
    self.group_indicator.y = self.y-10
```

Dette får agentens "gruppe-indikator" til at følge med den rundt.



Mere/mindre afstand

Prøv at variere afstand, agenterne holder, og den afstand, de kan smitte på.

For at afprøve virkningen af forskellige tiltag, gør vi nu sådan, at agenternes fysiske afstand og smitterækkevidde kan justeres, imens simulationen køres.

Tilføj to *sliders* til modellen med følgende kode (indsæt dem samme sted, som du laver knapper/checkboxes):

```
epidemic_model.add_controller_row()
epidemic_model.add_slider("social_distance", 50, 0, 80)
epidemic_model.add_controller_row()
epidemic_model.add_slider("infection_distance", 15, 0, 40)
```

Dette giver to sliders, som kan bruges til at justere variablene `social_distance` og `infection_distance`. De to første tal er minimums- og maksimumsværdierne, og det sidste tal er startværdien.

Ændr nu denne linje i `Person.step`:

```
for agent in self.agents_nearby(20):
```

til denne:

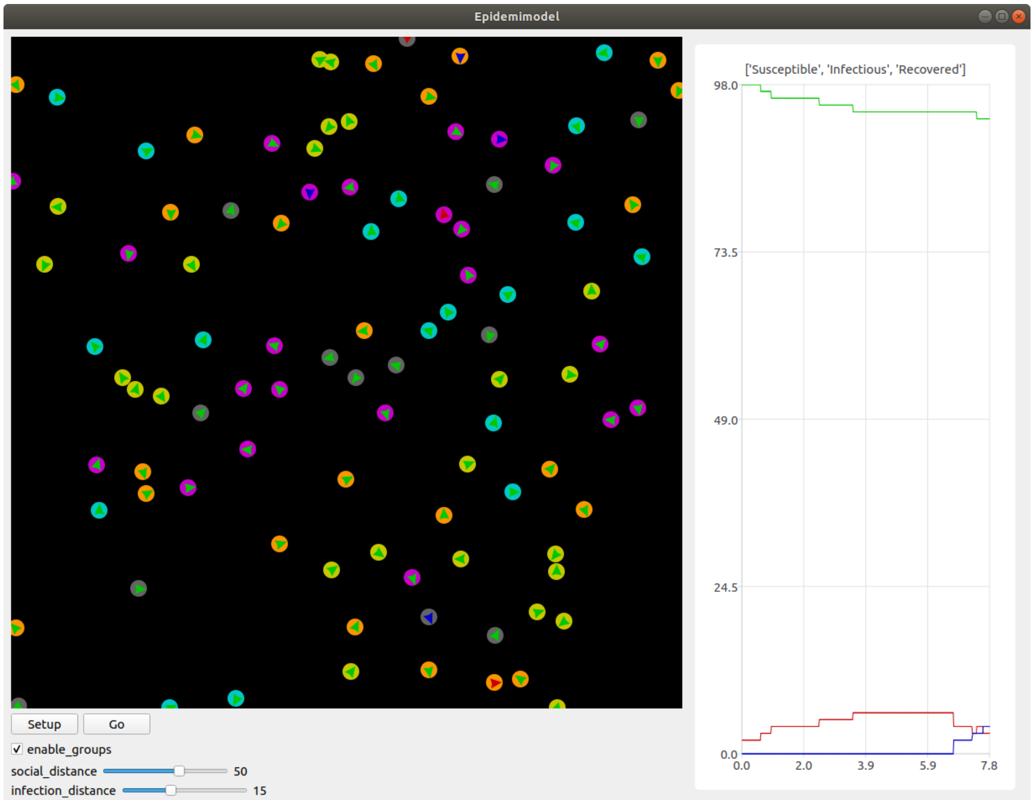
```
for agent in self.agents_nearby(model.social_distance):
```

og ændr denne:

```
for agent in self.agents_nearby(12):
```

til denne:

```
for agent in self.agents_nearby(model.infection_distance):
```



Prøv at køre simulationen, og juster på værdierne undervejs. Overvej, hvilken indflydelse forholdet mellem de to værdier har på smittetallene.

Samlet kode

Her er den samlede kode du gerne skulle have nu:

```
from agents import Model, Agent, run
from random import randint


class Person(Agent):
    def setup(self, model):
        model.Susceptible += 1
        self.category = 0
        self.color = (0, 200, 0)
        if randint(1, 50) == 1:
            self.infect(model)

        if model.enable_groups:
            self.group = randint(1, 5)
            self.group_indicator = model.add_ellipse(
                self.x - 10, self.y - 10, 20, 20, (0, 0, 0)
            )
            if self.group == 1:
                self.group_indicator.color = (200, 200, 0)
```

(continues on next page)

(continued from previous page)

```
            elif self.group == 2:
                self.group_indicator.color = (0, 200, 200)
            elif self.group == 3:
                self.group_indicator.color = (200, 0, 200)
            elif self.group == 4:
                self.group_indicator.color = (100, 100, 100)
            elif self.group == 5:
                self.group_indicator.color = (250, 150, 0)

    def step(self, model):
        if model.enable_groups:
            self.group_indicator.x = self.x - 10
            self.group_indicator.y = self.y - 10
        new_direction = 0
        nearby_agents = 0
        for agent in self.agents_nearby(model.social_distance):
            if model.enable_groups and agent.group != self.group:
                new_direction += self.direction_to(agent.x, agent.y)
                nearby_agents += 1
        if nearby_agents > 0:
            self.direction = (new_direction / nearby_agents) + 180
        else:
            self.direction += randint(-10, 10)
        self.forward()
        if self.category == 1:
            for agent in self.agents_nearby(model.infection_distance):
                if agent.category == 0:
                    agent.infect(model)
            self.infection_level -= 1
            if self.infection_level == 0:
                self.turn_immune(model)

    def infect(self, model):
        model.Susceptible -= 1
        model.Infectious += 1
        self.color = (200, 0, 0)
        self.category = 1
        self.infection_level = 600

    def turn_immune(self, model):
        model.Infectious -= 1
        model.Recovered += 1
        self.color = (0, 0, 200)
        self.category = 2


def model_setup(model):
    model.reset()
    model.Susceptible = 0
    model.Infectious = 0
    model.Recovered = 0
    for person in range(100):
        model.add_agent(Person())


def model_step(model):
    for person in model.agents:
```

(continues on next page)

(continued from previous page)

```
        person.step(model)
    model.update_plots()


epidemic_model = Model("Epidemimodel", 100, 100)

epidemic_model.add_button("Setup", model_setup)
epidemic_model.add_button("Go", model_step, toggle=True)
epidemic_model.line_chart(
    ["Susceptible", "Infectious", "Recovered"], [(0, 200, 0), (200, 0, 0), (0, 0,␣
↪200)]
)
epidemic_model.bar_chart(["Susceptible", "Infectious", "Recovered"], (200, 200, 200))
epidemic_model.add_checkbox("enable_groups")
epidemic_model.add_controller_row()
epidemic_model.add_slider("social_distance", 50, 0, 80)
epidemic_model.add_controller_row()
epidemic_model.add_slider("infection_distance", 15, 0, 40)

run(epidemic_model)
```

1.5.4 Del 4: Mutationer

Gode nyheder! Din model er blevet godt modtaget af regeringen, og de begynder snart at tage den i brug, for at vurdere, hvilke tiltag de skal sætte i værks. Pludselig bliver du dog ringet op af en forsker fra Statens Serum Institut, der fortæller dig, at din model er mangelfuld! De siger, at modellen mangler detaljer om, hvordan sygdommen kan *mutere* sig selv hen ad vejen. Forskeren giver dig en liste over ting, der skal tilføjes, og du skynder dig at gå i gang.

Virus-klasse

Fordi, at virussens opførsel bliver mere avanceret, er det nu nødvendigt at give den sin egen klasse, ligesom med `Person` klassen. Tilføj følgende klasse, oven over `Person` klassen:

```
class Virus():
    def __init__(self, mutation):
        self.infection_level = 600
        self.mutation = mutation

    def mutate(self):
        return Virus(self.mutation)
```

`infection_level` skal have samme funktionalitet som før. Vi kommer til at beskrive `mutation` senere.

Erstat nu denne kode i `Person.setup`:

```
if randint(1,50) == 1:
    self.infect(model)
```

med denne:

```
self.virus = None
if randint(1,50) == 1:
    self.infect(model, Virus(5))
```

I stedet for at agenten bare "simulerer" en virus ved at bruge sin `category` og `infection_level`, bærer den nu rundt på et *virus-objekt*, der holder styr på dette.

Dette betyder så også, at vi skal ændre alle de steder, der har noget at gøre med agentens infektion, til at bruge denne klasse i stedet. Ændr `Person.infect` til denne:

```
def infect(self, model):
    model.Susceptible -= 1
    model.Infectious += 1
    self.color = (200,0,0)
    self.category = 1
    self.virus = virus
```

og `Person.turn_immune` til denne:

```
def turn_immune(self, model):
    model.Infectious -= 1
    model.Recovered += 1
    self.color = (0,0,200)
    self.category = 2
    self.virus = None
```

Ændr til sidst dette stykke i `Person.step`:

```
if self.category == 1:
    for agent in self.agents_nearby(model.infection_distance):
        if agent.category == 0:
            agent.infect(model)
    self.infection_level -= 1
    if self.infection_level == 0:
        self.turn_immune(model)
```

til dette:

```
if self.category == 1:
    for agent in self.agents_nearby(model.infection_distance):
        if agent.category == 0:
            agent.infect(model, self.virus.mutate())
    self.virus.infection_level -= 1
    if self.virus.infection_level == 0:
        self.turn_immune(model)
```

Her inficerer vi altså den anden agent med et nyt virus-objekt lavet med `Virus.mutate`, fremfor "bare" at sætte dens `infection_level`.

Prøv at køre modellen, og se, om alt kører som det burde. Der burde der ikke være nogen forskel fra sidst.

Mutationsstadier

Hovedideen med at lave `Virus`-klassen er, at vi kan gemme information om dens *mutationsstadie* i den, fremfor at gemme den i agenten, der bærer den.

Vi vil nu ændre en smule i modellens opsætning. I stedet for, at der kun findes én variant af sygdommen, gør vi nu sådan, at sygdommen kan findes i *flere* varianter, og at man, hvis man har været smittet, kun bliver immun over for den variant, man har været smittet med.

Vi starter med at give agenten en liste over immuniteter. Tilføj denne linje til `Person.setup` inden, at agenten bliver tilfældigt inficeret:

```
self.immunities = []
```

Denne liste skal så indeholde alle de *mutations*-ID'er for de virusser, den har været smittet med. I den sammenhæng skal vi også checke, at agenten ikke bliver smittet med en immun virus, når den inficeres. I `Person.infect`, sæt alt koden ind i følgende `if`-sætning:

```
if not virus.mutation in self.immunities:
```

Så køres resten af koden ikke, hvis agenten allerede har været smittet med denne variation af virus.

Vi vil gerne have mulighed for at se med et øjekast, hvilken slags mutation, en agent er inficeret med. Ændr derfor denne linje i `Person.infect`:

```
self.color(200,0,0)
```

til denne:

```
self.color = (200,150-30*virus.mutation,150-30*virus.mutation)
```

Jo højere `Virus.mutation` er, jo mere rød farves agenten.

Samtidig ændrer vi nu lidt på `Person.turn_immune`, da agenterne i stedet bliver gradvist immune, fremfor at blive komplet immune efter første gang med sygdommen.

Erstat `Person.turn_immune` med nedenstående:

```
def turn_immune(self, model):
    model.Infectious -= 1
    model.Susceptible += 1
    self.color = (200-30*len(self.immunities),200,200-30*len(self.immunities))
    self.category = 0
    self.immunities.append(self.virus.mutation)
    self.virus = None
```

Der er nogle ændringer i forhold til den nuværende:

- I stedet for at sætte agentens kategori til 2, sætter vi den tilbage til 0, da agenten egentlig ikke bliver immun, men går tilbage til at være modtagelig. Af samme årsag lægger vi 1 til `model.Susceptible` i stedet for `model.Recovered`.
- Agentens farve bliver nu mere grøn, jo mere resistent den er (altså jo flere sygdomme den har haft).
- Vi tilføjer virussens "*mutation-ID*" til agentens liste over immuniteter. Den kan altså ikke smittes med denne mutation fremover.

Ændr i samme omgang også denne linje i `Person.setup`:

```
self.color = (0,200,0)
```

til denne:

```
self.color = (200,200,200)
```

Vi gør også sådan, at hvis en virus har muteret nok gange, kan den ikke længere smitte. Opdater `if`-sætningen i smittetrinet i `Person.step`, sådan at der i stedet for:

```
if agent.category == 0:
    agent.infect(model, self.virus.mutate())
```

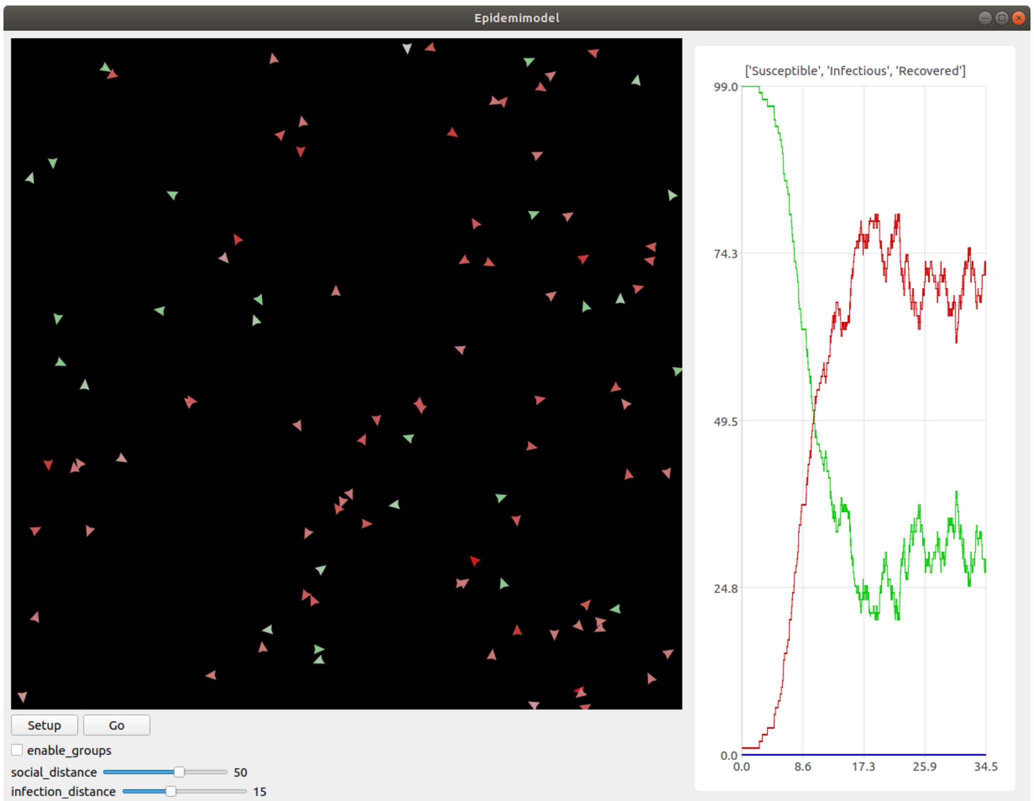
står:

```
if agent.category == 0 and self.virus.mutation > 0:
    agent.infect(model, self.virus.mutate())
```

Til sidst gør vi sådan, at der er en 25% chance for, at virussen muterer, når den spredes til en anden agent. Erstat `Virus.mutate` med:

```
def mutate(self):
    if randint(1,4) < 4:
        return Virus(self.mutation)
    else:
        return Virus(self.mutation-1)
```

Prøv at køre modellen nu, og observer grafen. Kan du se, hvordan de forskellige "bølger" af mutationer optræder?



Mutationseffekter

Lige nu har de forskellige mutationer ikke nogen egentlig forskel, ud over deres farve. Vi laver nu om på det, sådan at deres sygdomsperiode og infektionsradius ændres, når de muterer.

Vi gør dette ved at ændre på den måde, `Virus`-objektet oprettes på. Erstat `Virus.__init__` med følgende:

```
def __init__(self, mutation, duration, radius):
    self.mutation = mutation
    self.duration = duration
    self.radius = radius
    self.infection_level = self.duration
```

Dette gør, at vi kan specificere varigheden og rækkevidden for et virus-objekt, når vi laver det.

Ændr på samme måde `Virus.mutate` til følgende:

```
def mutate(self):
    if randint(1,4) < 4:
        return Virus(self.mutation,
                     self.duration,
                     self.radius)
    else:
        return Virus(self.mutation-1,
                     self.duration + randint(-100,100),
                     self.radius + randint(-5,5))
```

Her gør vi sådan, at virussens varighed og rækkevidde justeres en smule, når den muterer.

Når vi opretter en ny `Virus`, bliver vi så nødt til også at give en oprindelig værdi for varighed og rækkevidde. Ændr denne linje i `Person.setup`:

```
self.infect(model, Virus(5))
```

til denne:

```
self.infect(model, Virus(5, 600, model.infection_distance))
```

Til sidst, ændr denne linje i `Person.step`:

```
for agent in self.agents_nearby(model.infection_distance):
```

til denne:

```
for agent in self.agents_nearby(self.virus.distance):
```

Prøv at køre modellen og se, om du ser en mærkbar forskel.

Samlet kode

Her er den samlede kode du gerne skulle have nu:

```
from agents import Model, Agent, run
from random import randint


class Virus:
    def __init__(self, infection_level, mutation):
        self.infection_level = infection_level
        self.mutation = mutation


class Person(Agent):
    def setup(self, model):
        model.Susceptible += 1
        self.category = 0
        self.color = (200, 200, 200)

        self.immunities = []
        self.virus = None

        if randint(1, 50) == 1:
```

(continues on next page)

(continued from previous page)

```
            self.infect(model, Virus(600, 5))

        if model.enable_groups:
            self.group = randint(1, 5)
            self.group_indicator = model.add_ellipse(
                self.x - 10, self.y - 10, 20, 20, (0, 0, 0)
            )
            if self.group == 1:
                self.group_indicator.color = (200, 200, 0)
            elif self.group == 2:
                self.group_indicator.color = (0, 200, 200)
            elif self.group == 3:
                self.group_indicator.color = (200, 0, 200)
            elif self.group == 4:
                self.group_indicator.color = (100, 100, 100)
            elif self.group == 5:
                self.group_indicator.color = (250, 150, 0)

    def step(self, model):
        if model.enable_groups:
            self.group_indicator.x = self.x - 10
            self.group_indicator.y = self.y - 10
        new_direction = 0
        nearby_agents = 0
        for agent in self.agents_nearby(model.social_distance):
            if model.enable_groups and agent.group != self.group:
                new_direction += self.direction_to(agent.x, agent.y)
                nearby_agents += 1
        if nearby_agents > 0:
            self.direction = (new_direction / nearby_agents) + 180
        else:
            self.direction += randint(-10, 10)
        self.forward()
        if self.category == 1:
            for agent in self.agents_nearby(model.infection_distance):
                if agent.category == 0 and self.virus.mutation > 0:
                    agent.infect(
                        model, Virus(600, self.virus.mutation - randint(0, 1))
                    )
            self.virus.infection_level -= 1
            if self.virus.infection_level == 0:
                self.turn_immune(model)

    def infect(self, model, virus):
        if virus.mutation not in self.immunities:
            model.Susceptible -= 1
            model.Infectious += 1
            self.color = (
                200,
                150 - 30 * virus.mutation,
                150 - 30 * virus.mutation,
            )
            self.category = 1
            self.virus = virus

    def turn_immune(self, model):
        model.Infectious -= 1
```

(continues on next page)

(continued from previous page)

```
        model.Susceptible += 1
        self.color = (
            200 - 30 * len(self.immunities),
            200,
            200 - 30 * len(self.immunities),
        )
        self.category = 0
        self.immunities.append(self.virus.mutation)
        self.virus = None


def model_setup(model):
    model.reset()
    model.Susceptible = 0
    model.Infectious = 0
    model.Recovered = 0
    for person in range(100):
        model.add_agent(Person())


def model_step(model):
    for person in model.agents:
        person.step(model)
    model.update_plots()


epidemic_model = Model("Epidemimodel", 100, 100)

epidemic_model.add_button("Setup", model_setup)
epidemic_model.add_button("Go", model_step, toggle=True)
epidemic_model.line_chart(["Susceptible", "Infectious"], [(0, 200, 0), (200, 0, 0)])
epidemic_model.bar_chart(["Susceptible", "Infectious"], (200, 200, 200))
epidemic_model.add_checkbox("enable_groups")
epidemic_model.add_controller_row()
epidemic_model.add_slider("social_distance", 50, 0, 80)
epidemic_model.add_controller_row()
epidemic_model.add_slider("infection_distance", 15, 0, 40)

run(epidemic_model)
```

1.6 Tutorial: Simpelt økosystem med rovdyr og byttedyr

Vi vil nu lave en lille model med AgentsPy. Vi laver en såkaldt *predator-prey-model*, altså en model med rovdyr og byttedyr.

Start med at lave en ny python-fil, *prey.py*, og skriv følgende:

```
from agents import *

model = Model("Predator-prey-model", 50, 50)

run(model)
```

Dette laver en model med 50x50 felter. Hvis du kører scriptet, bør du få et vindue med en sort firkant.

Vi starter med at lave en `Prey` klasse til vores byttedyr. Lav den på følgende måde:

```
class Prey(Agent):
    def setup(self, model):
        pass

    def step(self, model):
        self.direction += randint(-10,10)
        self.forward()
```

Den skal altså ved hvert trin ("step") ændre sin retning lidt, og bevæge sig fremad.

Lav nu en `model_setup` funktion, der "genstarter" modellen og tilføjer 100 nye `Prey` agenter:

```
def model_setup(model):
    model.reset()
    for a in range(100):
        model.add_agent(Prey())
```

Tilføj så en *Setup* knap til modellen, der kører `model_setup` funktionen:

```
model.add_button("Setup", model_setup)
```

Prøv at køre scriptet nu, og se, hvad der sker. Du burde have en *Setup* knap, der laver 100 agenter, når den klikkes på.

Vi får nu agenterne til at bevæge sig. Tilføj en `model_step` funktion, der får byttedyrene til at køre deres egen `step` funktion:

```
def model_step(model):
    for a in model.agents:
        a.step(model)
```

Lav nu en *Go* knap, som kan slås til og fra, og som konstant kører `model_step` funktionen, når den er slået til:

```
model.add_toggle_button("Go", model_step)
```

Nu har vi vores grundlæggende model. Vi vil nu gøre det muligt for byttedyrene at spise græs, og formere sig, hvis de har spist nok græs.

Vi starter med at tilføje græs. Tilføj i `model_setup`:

```
for t in model.tiles:
     t.info["grass"] = True
     t.color = (0, 150, 0)
```

Dette gør sådan, at alle felter starter med at være indikeret som græs. For at de bliver opdateret med en mere "jordlignende" farve, når græsset bliver spist, tilføj følgende i `step`:

```
for t in model.tiles:
    if t.info["grass"]:
        t.color = (0, 150, 0)
    else:
        t.color = (80, 80, 0)
        if randint(1, 500) == 500:
            t.info["grass"] = True
```

Felter, der har `info["grass"] = True` bliver nu farvet grønne, imens dem der har `info["grass"] = False`, bliver farvet brune. Felter, der mangler græs, har desuden hvert step en chance for, at deres græs vokser tilbage igen.

Vi gør nu sådan, at byttedyr kan spise græs, formere sig, hvis de spiser nok, og dø, hvis de ikke får nok at spise. Vi laver først funktionaliteten for at spise. Tilføj i `Prey` klassens `setup` funktion:

```
self.food = 0
self.time_since_eating = 0
self.color = (100,100,250)
```

Vi giver dem en blå farve, så vi kan adskille dem fra de rovdyr, vi senere tilføjer.

Tilføj derefter i `Prey` klassens `step` funktion:

```
tile = self.current_tile()
if tile.info["grass"]:
    self.food += 1
    self.time_since_eating = 0
    tile.info["grass"] = False
if self.food > 10:
    new_prey = Prey()
    new_prey.x = self.x
    new_prey.y = self.y
    model.add_agent(new_prey)
    self.food = 0
self.time_since_eating += 1
if self.time_since_eating > 60:
    self.destroy()
```

Her gør byttedyret følgende: * Hvis den står på et felt med græs, spis græsset og læg 1 til "mad-tælleren". * Hvis den har spist nok græs, lav et nyt byttedyr og sæt "mad-tælleren" til 0. * Hvis der er gået for lang tid siden den sidst har spist, destruerer den sig selv.

Vi vil gerne gøre det muligt at indstille undervejs i modellen, hvor meget græs, et byttedyr skal spise, før det kan formere sig, og hvor lang tid dyret skal gå uden mad, før at det dør.

I `model_setup`, tilføj disse to linjer:

```
model.reproduce_food_count = 10
model.max_time_since_eating = 60
```

Erstat så følgende linjer i `Prey` klassens `step` funktion:

```
if self.food > 10:
...
if self.time_since_eating > 60:
```

med disse

```
if self.food > model.reproduce_food_count:
..
if self.time_since_eating > model.max_time_since_eating:
```

Tilføj så to justerbare *sliders* ved at indsætte disse to linjer kode, efter at knapperne tilføjes:

```
model.add_slider("reproduce_food_count", 10, 1, 30)
model.add_slider("max_time_since_eating", 60, 10, 120)
```

Nu er vores byttedyr færdigt.

Man kan nu, hvis man vil, tilføje *rovdyr* til simuleringen. Man kan bruge følgende klasse som udgangspunkt:

```
class Predator(Agent):
    def setup(model):
        self.size = 15
        self.color = (150,0,0)

    def step(model):
        self.direction += randint(-10,10)
        self.forward()
```

Rovdyret bør have følgende funktionalitet:

- Hvis der er et byttedyr på samme felt som rovdyret, skal det spises (brug en kombination af `Agent.current_tile()` og `Tile.get_agents()` til at finde ud af, om der er et byttedyr på samme felt).
- Hvis rovdyret har spist nok byttedyr, skal det formere sig (brug samme fremgangsmåde som for byttedyret, der spiser græs).
- Hvis rovdyret ikke har spist noget i lang nok tid, skal det dø (brug også her samme fremgangsmåde som for byttedyret).

CHAPTER 2

Brugerguide

2.1 Overblik over API'et

AgentsPy er et bibliotek, der tilbyder værktøjer, som kan bruges til agent-baseret modellering. Agent-baseret modellering fungerer ved først at opbygge et miljø med agenter, der har en prædefineret opførsel, og så simulere hele systemet baseret på agenternes opførsel. Modellen kan så vise, hvordan agenterne interagerer med hinanden, og systemet som helhed. I AgentsPy kaldes en agent passende for Agent, et "felt" i miljøet kaldes en Tile, og miljøet som helhed kaldes en Model.

For at give agenterne deres prædefinerede opførsel, skal man kode dem. Dette gøres typisk ved først at definere en klasse, som nedarver fra Agent, for eksempel Person i epidemimodellen. Heri kan man så definere en step funktion, der beskriver, hvordan agenten opfører sig i et enkelt simulationstrin. Et eksempel kunne være:

```
def step(model):
    self.forward()
```

hvilket rykker agenten et skridt fremad ved hver trin.

Man kan så lave en model_step funktion, der simulerer alle agenterne i en model:

```
def model_step(model):
    for a in model.agents:
            a.step(model)
```

Miljøet ændres oftest som en direkte konsekvens af agenternes opførsel. Man kan for eksempel vælge at farve de felter, en agent har besøgt, røde:

```
def step(model):
    self.forward()
        t = self.current_tile()
        t.color (255, 0, 0)
```

AgentsPy tilbyder også muligheden for at justere på simulationen undervejs. Man kan for eksempel bruge en knap til at starte og stoppe simulationen:

```
model.add_toggle_button("Go", model_step)
```

De funktioner, der kan bruges til at tilføje elementer til kontrolpanelet, er:

- add_button: Tilføjer en knap, der kan klikkes på gentagne gange.
- add_toggle_button: Tilføjer en knap, der kan slås til og fra.
- add_slider: Tilføjer en bevægelig knap, der kan bruges til at justere værdien af en numerisk variabel i modellen.
- add_checkbox: Tilføjer et afkrydsningsfelt, der kan bruges til at justere værdien af en sandhedsvariabel i modellen.

Ud over funktioner til kontrolpanelet, er der også funktioner som tilføjer forskellige slags grafer til modellen, som løbende viser data, herunder:

- line_chart: Tilføjer en graf med en eller flere linjer, der beskriver modellens variable over tid.
- bar_chart: Tilføjer et søjlediagram, der viser værdien af modellens variable i øjeblikket.
- histogram: Tilføjer et histogram, der viser, hvordan en bestemt variabel for alle agenterne fordeler sig i givne intervaller.
- agent_line_chart: Tilføjer en graf med en linje for hver agent, der viser værdien af denne variabel over tid.
- monitor: Tilføjer et lille felt til kontrolpanelet, der viser værdien for en enkelt variabel i modellen.

2.2 Agent

Agents are the units that make up the "active" portion of the model. They generally move around the model area, interacting with each other and the area itself.

class agents.**Agent**

Creates an agent with a random position, direction and color. Has no initial model; this must be provided by Agent.set_model.

agents_nearby(*distance*, *agent_type=None*)

Returns a list of nearby agents. May take a type as argument and only return agents of that type.

Parameters

- **distance** – The radius around the agent to search in.
- **agent_type** – If provided, only returns agents of this type.

backward(*distance=None*)

Moves the agent in the opposite direction of its current orientation.

Parameters Distance – The distance to move the agent. If none is specified, it moves a distance equal to its speed-attribute.

center_in_tile()

Move the agent to the center of the tile it is standing on.

color

The color of the agent. Must be provided as an RGB 3-tuple, e.g. (255, 255, 255) to color the agent white.

current_tile()

Returns the tile that the agent is currently standing on, based on its coordinates.

The tile returned is the one that overlaps with the exact center of the agent, so even if the agent visually covers multiple tiles due to its size, only one tile is returned.

destroy()

Marks the agent for destruction, removing it from the set of agents in the model.

direction

The direction of the agent, measured in degrees.

direction_to(*other_x*, *other_y*)

Calculate the direction in degrees from the agent to a given point.

Parameters

- **other_x** – The x-coordinate of the target point.
- **other_y** – The y-coordinate of the target point.

distance_to(*other_x*, *other_y*)

Returns the distance between the agent and another point.

Parameters

- **other_x** – The x-coordinate of the target point.
- **other_y** – The y-coordinate of the target point.

forward(*distance=None*)

Moves the agent forward in the direction it is currently facing.

Parameters Distance – The distance to move the agent. If none is specified, it moves a distance equal to its speed-attribute.

is_destroyed()

Returns True or False whether or not the agent is destroyed.

jump_to(*x*, *y*)

Move the agent to a specified point.

Parameters

- **x** – Destination x-coordinate.
- **y** – Destination y-coordinate.

jump_to_tile(*t*)

Move the agent to the center of a specified tile.

Parameters t – Destination tile.

nearby_tiles(*x1*, *y1*, *x2*, *y2*)

Returns a rectangle of tiles relative to the agent's current position.

Parameters

- **x1** – The x-coordinate of the top-left tile (relative to the current tile).
- **y1** – The y-coordinate of the top-left tile (relative to the current tile).
- **x2** – The x-coordinate of the bottom-right tile (relative to the current tile).
- **y2** – The y-coordinate of the bottom-right tile (relative to the current tile).

neighbor_tiles()

Returns the surrounding tiles as a 3x3 grid. Includes the current tile.

point_towards (*other_x*, *other_y*)
Make the agent orient itself towards a given point.

Parameters

- **other_x** – The x-coordinate of the target point.
- **other_y** – The y-coordinate of the target point.

rotate (*degrees*)
Make the agent turn the given number of degrees. Positive is counter-clockwise, negative is clockwise.

Parameters degrees – The amount of degrees to turn.

set_model (*model*)
Provides the Model object that the agents belongs to.

The stored model is used in other methods such as `Agent.agents_nearby` and `Agent.current_tile`, which rely on information about other objects in the model.

Parameters model – The model to assign the agent to.

setup (*model*)
This method is run when the agent is added to a model. The method is empty by default, and is intented to be overwritten by a subclass.

Parameters model – The model object that the agent has been added to.

shape
The shape of the agent.

size
The size of the agent. For an agent with the circle shape, this corresponds to its radius.

update_current_tile ()
Updates the tile that the agent is currently standing on.

Effectively, this removes the agent from the set of agents standing on the previous tile, and adds it to the set of agents standing on the current tile.

2.3 Tile

A square, of which many make up the "floor" of the model.

class `agents.`**Tile** (*x*, *y*, *model*)
Creates a tile. *x* and *y* is the tile's position in the *tile grid*, not absolute coordinates for the model.

Parameters

- **x** – The tile's x-coordinate in the tile grid.
- **y** – The tile's y-cooridnate in the tile grid.
- **model** – The model that the tile is a part of.

add_agent (*agent*)
Adds an Agent to the set of agents standing on the tile. Usually called by the method `Agent.update_current_tile`.

Parameters agent – The agent to add.

color
The color of the agent. Must be provided as an RGB 3-tuple, e.g. (255, 255, 255) to color the agent white.

get_agents()
Gets the set of agents currently on the tile.

remove_agent(*agent*)
Removes an Agent `agent` from the set of agents standing on the tile. Usually called by the method `Agent.update_current_tile`.

Parameters **agent** – The agent to remove.

2.4 Model

Models contain the agents and tiles that make up the simulation. They also provide some functionality for manipulating said simulation, such as buttons, and ways to visualize simulation data, such as graphs.

To run a model, use

```
agents.run(model)
```

class agents.**Model**(*title*, *x_tiles=50*, *y_tiles=50*, *tile_size=8*, *cell_data_file=None*)
Creates a model with the given title. There are two ways of creating a model; creating a blank model of size *x_tiles* times *y_tiles* with no predetermined data, or creating a model with predetermined data from a *cell_data_fille*.

Parameters

- **title** – The title of the model (to show in the simulation window).
- **x_tiles** – The number of tiles on the x-axis. Ignored if a *cell_data_file* is provided.
- **y_tiles** – The number of tiles on the y-axis. Ignored if a *cell_data_file* is provided.
- **tile_size** – The width/height of each tile in pixels.
- **cell_data_file** – If provided, generates a model from the data file instead. The data is not immediately to the tiles, but must be applied with `Model.reload()`.

add_agent(*agent*, *setup=True*)
Adds an agent to the model.

Parameters

- **agent** – The agent to add to the model.
- **setup** – Whether or not to run the agent's `setup` function (default `True`).

add_agents(*agents*)
Adds a collection of agents.

Parameters **agents** – The agents to add.

add_button(*label*, *func*, *toggle=False*)
Adds a button that runs a provided function when pressed. Can be specified to be a toggled button, which will cause the button to continuously call the function while toggled on.

Parameters

- **label** – The label on the button.
- **func** – The function to run when the button is pressed.
- **toggle** – Whether or not the button should be a toggled button.

add_checkbox (*variable*)
Adds a checkbox that can be used to change the value of a variable between true and false.

Parameters **variable** – The name of the variable to adjust. Must be provided as a string.

add_controller_row ()
Creates a new row to place controller widgets on (buttons, sliders, etc.).

add_ellipse (*x*, *y*, *w*, *h*, *color*)
Draws an ellipse on the simulation area. Returns a shape object that can be used to refer to the ellipse.

Parameters

- **x** – The top-left x-coordinate of the ellipse.
- **y** – The top-left y-coordinate of the ellipse.
- **w** – The width of the ellipse.
- **h** – The height of the ellipse.
- **color** – The color of the ellipse.

add_rect (*x*, *y*, *w*, *h*, *color*)
Draws a square on the simulation area. Returns a shape object that can be used to refer to the square.

Parameters

- **x** – The top-left x-coordinate of the square.
- **y** – The top-left y-coordinate of the square.
- **w** – The width of the square.
- **h** – The height of the square.
- **color** – The color of the square.

add_slider (*variable*, *initial*, *minval=0*, *maxval=100*)
Adds a slider that can be used to adjust the value of a variable in the model.

Parameters

- **variable** – The name of the variable to adjust. Must be privded as a string.
- **minval** – The minimum value of the variable.
- **maxval** – The maximum value of the variable.
- **initial** – The initial value of the variable.

agent_line_chart (*variable*, *min_y=None*, *max_y=None*)
Adds a line chart to the simulation window that shows the trend of multiple variables over time.

Parameters

- **variables** – The names of the variables. Must be provided as a list of strings.
- **colors** – The color of each line.
- **min_y** – The minimum value on the y-axis.
- **max_y** – The maximum value on the y-axis.

agents_ordered (*variable*, *increasing=True*)
Returns a list of agents in the model, ordered based on one of their attributes. Agents who do not have the attribute are not included in the list.

Parameters

- **variable** – The attribute to order by.
- **increasing** – Whether or not to order the agents in increasing or decreasing order (default `True`).

bar_chart(*variables*, *color*)
Adds a bar chart to the simulation window that shows the relation between multiple variables.

Parameters

- **variables** – The list of the variables. Must be provided as a list of strings.
- **color** – The color of all the bars.

clear_plots()
Clears the data from all plots.

clear_shapes()
Clears all shapes in the model.

disable_wrapping()
Disables wrapping. Agents attempting to move outside the simulation area will collide with the border and be moved back to the closest point inside.

enable_wrapping()
Enables wrapping, i.e. turns the simulation area *toroidal*. Agents exiting the simulation area on one side will enter on the other side.

get_shapes()
Returns an iterator containing all the shapes in the model.

histogram(*variable*, *minimum*, *maximum*, *bins*, *color*)
Adds a histogram to the simulation window that shows how the agents in the model are distributed based on a specific attribute.

Parameters

- **variable** – The name of the attribute to base the distribution on. Must be provided as a string.
- **minimum** – The minimum value of the distribution.
- **maximum** – The maximum value of the distribution.
- **bins** – The number of bins in the histogram.
- **color** – The color of all the bars.

is_paused()
Returns whether the model is paused or not.

line_chart(*variables*, *colors*, *min_y=None*, *max_y=None*)
Adds a line chart to the simulation window that shows the trend of multiple variables over time.

Parameters

- **variables** – The names of the variables. Must be provided as a list of strings.
- **colors** – The color of each line.
- **min_y** – The minimum value on the y-axis.
- **max_y** – The maximum value on the y-axis.

monitor(*variable*)
Adds a single line that shows the value of the given variable.

Parameters **variable** – The variable to monitor.

on_close(*func*)
Defines a function to be run when the simulation window is closed. This is generally used to close any open file pointers.

pause()
Pauses the model. The main effect of this is to ignore the "on" status of any toggled buttons, meaning that *step* functions and similar are not run.

reload()
Applies the data from the cell-data-file to the tiles in the model. Only usable if the model was created with a `cell_data_file` in the constructor.

reset()
Resets the model by doing the following:

- Destroys all agents.
- Clears the set of agents.
- Clears the set of shapes.
- Clears all tiles (removes all of their `info` and colors them black).
- Clears all plots.
- Unpauses the model.

tile(*x*, *y*)
Returns the tile at the (x,y) position in the tile-grid (*not* the (x,y) position of the simulation area).

Parameters

- **x** – The x-coordinate of the tile.
- **y** – The y-coordinate of the tile.

unpause()
Unpauses the model. See *Model.pause()*.

update_plots()
Updates all plots with the relevant data. Usually called in each iteration of the simulation (i.e. in a `step` function or similar).

wrapping()
Returns whether wrapping is enabled or not.

2.4.1 Cell-data file format

The two initial lines are skipped, and may be used to describe the cell-data or for other comments. The third line must contain a set of column names, seperated by tabs. The column names specify the names of variables stored by each tile. The first and second column cannot be used for variables, but must instead contain x and y coordinates, respectively.

The remaining lines of the file should then contain the coordinates and variable data. The start of a cell-data file might look like this:

```
This file contains cell data for a model where each cell/tile has some resource.
The data specifies the rate of resource production and max resource content for each␣
↪cell.
x    y       prod    max_res
```

(continues on next page)

(continued from previous page)

```
0    0        0.15    10.0
1    0        0.20    20.0
2    0        0.05    30.0
3    0        0.35    5.0
...
```

2.5 SimpleModel

2.5.1 Description

To simplify the generation of a model, one may use a `SimpleModel` object instead of a `Model`. The main difference is that the `SimpleModel` constructor also takes a `setup` and `step` function as arguments, and then generates the appropriate **Setup** and **Go** buttons automatically, rather than having the user do it manually.

The `SimpleModel` also provides an error message if the **Go** button is pressed before the **Setup** button.

2.5.2 Fields

See *Model*.

2.5.3 Methods

- **`__init__(title, x_tiles, y_tiles, setup_func, step_func, tile_size=8)`**
  Creates a model with the given title and number of tiles on the x and y axis, as well as tile size. Also adds **Setup** and **Go** buttons to the simulation area.

For the remaining methods, see *Model*.

CHAPTER 3

Om projektet

Målet med *AgentsPy* er at gøre det ligeså nemt at arbejde med agent-baseret simulering som i NetLogo, uden at eleverne skal lære et separat programmeringssprog. Biblioteket er stadig under udvikling på Datalogisk Institut, Københavns Universitet.

CHAPTER 4

Kan du ikke finde det du leder efter?

Prøv at slå op i registeret eller søg i dokumentationen:

- Søg i dokumentationen
- genindex over alle funktioner og klasser i AgentsPy

CHAPTER 5

License

AgentsPy er udgivet under GPLv3 licensen. Valget af licens stammer fra vores brug af PyQt5, som også er udgivet under GPLv3.

Dokumentationen er udgivet under Creative Commons BY-SA 4.0, undtagen enkelte figurer hvor vi har angivet original kildehenvisning

5.1 License

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling.

In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your

option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <https://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <https://www.gnu.org/licenses/why-not-lgpl.html>.

Index

S

T

U

W