Introduktion til brug af agents.py

<jkl@di.ku.dk>

28. juli 2020

1 Minerbots

Vi vil lave en simulation af nogle robotter på en fjern planet, der indsamler mineraler og skal prøve at undgå rumvæsener.

Lav en fil kaldet `minerbots.py`.
Begynd med først at indlæse biblioteket `agents`. Det gøres ved, øverst i filen, at skrive

```
from agents import *
```

1.1 Basics

Til at begynde med, skal vi først lave en model. Det gøres simpelt nok ved at skrive

```
miner_model = Model("MinerBots",100,100)
```

Her opretter vi en model kaldet `miner_model`, og definerer, at den skal have $100 * 100$ felter. For at køre modellen kan vi derefter skrive:

```
run(miner_model)
```

Køres `minerbots.py` burde et vindue med navnet `"MinerBots"` blive vist.

1.2 Setup

Vi vil gerne gøre det nemt at starte og genstarte modellen, når vi skal køre den. Derfor laver vi en *setup*-knap. Lav først en funktion der hedder `setup`, der tager en model som argument, og som ser således ud:

```
def setup(model):
    model.reset()
```

Indtil videre skal den bare "resette" modellen, altså fjerne alle agenter og nulstille alle felter (selvom der ingen er af dem endnu).
Tilføj nu følgende linje kode mellem oprettelsen af modellen og `run`-funktionen:

```
miner_model.add_button("Setup", setup)
```

Modellen burde nu have en ekstra "Setup"knap (der ikke gør noget, når man klikker på den).

1.3 Felter

Vi starter modellen med at generere den "planetoverflade", som vores robotter skal gå rundt på. De fleste af modellens felter skal være tom jord (rødbrun), men enkelte felter skal indeholde mineraler (lyseblå).

Vi kan generere modellens felter (kaldet *tiles*) ved at indsætte følgende kode i `setup` funktionen:

```
for t in model.tiles:
  if RNG(50) == 50:
      t.color = (0,255,255)
      t.info["has_mineral"] = True
  else:
      t.color = (200,100,0)
      t.info["has_mineral"] = False
```

`RNG` er en funktion, der giver et tilfældigt tal fra 0 til og med det givne tal, her 100. `color`-attributten bestemmer feltets farve. `info` er en *dictionary*, hvor man under bestemte nøgleord (her "`has_mineral`") kan gemme bestemte værdier (her `True/False`). Hvert felt har således hver sin dictionary.

1.4 De første robotter

Modellen er ikke særlig interessant uden nogle *agenter*. Til at starte med skal vi lave en klasse for vores robotter. Kald denne klasse `MinerBot`, og sørg for at den nedarver fra `Agent` på følgende måde:

```
class Minerbot(Agent):
```

Giv den en funktion, der hedder `setup`, som gør følgende:

```
def setup(self, model):
    self.size = 10
    self.color = (100, 100, 100)
```

Specielt for agenternes `setup`-funktion er, at den køres automatisk, når agenten tilføjes til modellen.
Man kan nu tilføje 10 robotter til modellen således (indsæt koden i `setup`):

```
bots = set([Minerbot() for _ in range(10)])
miner_model.add_agents(bots)
```

Kører man modellen, burde der gerne ses 10 grå cirkler - robotterne - spredt omkring landskabet.

1.5 Mere avancerede robotter

Robotterne er ikke særligt interessante, som de er nu. Vi starter med at gøre sådan, at de bare bevæger sig tilfældigt rundt. Tilføj først i `Minerbot`'s `setup` funktion følgende linje:

```
self.direction = RNG(360)
```

Dette får dem til at vælge en tilfældig retning, når de oprettes.
Tilføj nu en `step` funktion til `Minerbot`, der ligesom `setup` også tager en model som argument. Giv den følgende indhold:

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

Dette får agenterne til at bevæge sig fremad, når deres `step`-funktion kaldes.
Lav nu *udenfor* `Minerbot` en `step` funktion, der også tager en model som argument. Giv den følgende indhold:

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

Til sidst, tilføj ligesom med "Setup"knappen en "Go-knap, der kører `step`-funktionen:

```
miner_model.add_toggle_button("Go", step)
```

1.6 Endnu mere avancerede robotter

Robotterne gør noget nu, men de er stadig ikke særligt interessante. Vi gør nu sådan, at robotternes bevægelse bliver mere avanceret, og at de kan samle mineraler op.

For at gøre deres bevægelse til noget andet end bare en lige linje, gør vi sådan, at de ved hvert trin ændrer en lille smule på deres retning. Til `Minerbots.step`, tilføj

```
self.direction += RNG(20)-10
```

Robotterne burde nu bevæge sig mere naturligt rundt.

Vi vil i samme omgang også gøre det muligt for robotterne at samle mineraler op, hvis de står på et mineral-felt, og ikke allerede har en ladning mineraler.
Tilføj først til `Minerbots.setup`:

```
self.loaded = False
```

Vi bruger `loaded` til at indikere, om robotten allerede har en ladning af mineraler.
Tilføj yderligere til `Minerbots.step`:

```
t = self.current_tile()
    if t.info["has_mineral"] and not self.loaded:
    t.info["has_mineral"] = False
    t.color = (200,100,0)
    self.color = (100,100,255)
    self.loaded = True
```

1.7 Flere agenttyper

Vi giver nu robotterne et sted, hvor de kan læsse mineraler af.
Lav en ny klasse, der også nedarver fra `Agent`, kaldet `Homebase`. Vi vil gerne

have basen til at være en større grå cirkel i midten af simulationsområdet. Giv den følgende `setup`-funktion:

```
def setup(self, model):
    self.size = 20
    self.color = (200,200,200)
    self.x = model.width/2
    self.y = model.height/2
```

I samme omgang gør vi også sådan, at robotterne starter i basen. I `Minerbots.setup`, tilføj

```
self.x = model.width/2
self.y = model.height/2
```

Vi vil også gerne gøre det muligt for agenterne at læsse mineraler af. Tilføj en `step` funktion til `Homebase`, og giv den følgende indhold:

```
def step(self,model):
    for a in self.agents_nearby(self.size+5):
        if type(a) == Minerbot and a.loaded:
            a.loaded = False
            a.color = (100,100,100)
            self.size += 1
```

Funktionen tjekker, om der er robotter med last indenfor en vis afstand af basen, og, hvis, der er, tømmer den lasten og øger basens egen størrelse.

Vi gør også robotternes navigation lidt smartere. I `Minerbot.step`, erstat

```
self.forward()
self.direction += RNG(20)-10
```

med

```
if self.loaded:
    self.point_towards(model.width/2,model.height/2)
else:
    self.direction += RNG(20)-10
self.forward()
```

Til sidst skal vi tilføje en enkelt `Homebase` til modellen ved at indsætte

```
model.add_agent(Homebase())
```

i `step`-funktionen hvor robotterne også oprettes.

1.8 Grafer

Biblioteket gør det også muligt at tegne grafer over bestemte værdier i modellen. Start med, i modellens `setup` funktion, at skrive

```
model.clear_plots()
model["minerals_collected"] = 0
```

`Model`-klassen fungerer også som dictionary, så derfor er det også muligt at gemme værdier for bestemte nøgleord her.

Opdater så `Homebase.step` til at øge `minerals_collected`:

```
def step(self,model):
    for a in self.agents_nearby(self.size/2+5):
        if type(a) == Minerbot and a.loaded:
            a.loaded = False
            a.color = (100,100,100)
            self.size += 1
            model["minerals_collected"] += 1
```

Nu, hvor vi har en variabel der kan måles, kan vi lave en graf som viser dens ændring over tid.
For at gøre dette, skal vi først indikere i `step`-funktion, at denne skal opdatere grafen. Indsæt derfor

```
model.update_plots()
```

i `step`-funktionen. Nu er det bare at tilføje

```
miner_model.graph("minerals_collected",(0,255,255))
```

inden at modellen køres med `run`.

1.9 Fjendtlige agenter

For at øge spændingen lidt introducerer vi nu nogle fjendtlige rumvæsener, der prøver at fange og dræbe robotterne.

Begynd med at lave en ny `Alien`-klasse. I `Alien.setup` skal de gives en `size` på 15, en tilfældig `direction` (brug `RNG(360)`) og en `color` på $(255, 0, 0)$.
Lav en funktion `Alien.destroy_robot` med følgende indhold:

```
def destroy_robot(self):
    t = self.current_tile()
    for other in t.get_agents():
        if type(other) == Minerbot:
            other.destroy()
```

Lav så `Alien.step` med

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

Sidste trin er at tilføje tre `Alien` agenter, på samme måde som `Minerbot`s bliver tilføjet.

1.10 Styr robotterne

Vi gør nu sådan, at det er muligt for basen at producere en ny robot, men til gengæld mindske sin størrelse. Dette skal brugeren selv kunne styre ved hjælp af en knap.

Erstat først denne linje i `setup`:

```
model.add_agent(Homebase())
```

med disse:

```
model["Homebase"] = Homebase()
model.add_agent(model["Homebase"])
```

Idéen er, at vi så får mulighed for at referere til base-agenten ved at bruge `model["Homebase"]`, i stedet for at man hver gang skal iterere gennem `model.agents` for at finde den.

Lav nu en funktion `build_bot` med følgende udseende:

```
def build_bot(model):
    if model["Homebase"].size > 30:
        model["Homebase"].size -= 10
        model.add_agent(Minerbot())
```

Lav så en knap, der kører funktionen:

```
miner_model.add_button("Build new bot", build_bot)
```

Vi vil også gerne gøre det muligt at justere robotternes hastighed. Start med nederst i `setup` at tilføje:

```
model["robot_speed"] = 1
```

Tilføj så, øverst i `Minerbot.step`:

```
self.speed = model["robot_speed"]
```

Lav til sidst en *slider*, der kan styre værdien af `"robot_speed"`:

```
miner_model.add_slider("robot_speed", 1, 5, 2)
```

hvor 1 er minimum, 5 er maksimum, og 2 er startværdien.