Agusti Conesa Gago, Enric Segura Sánchez

SIMULACIÓ BASADA EN AGENTS

Estudi del framework: Mesa

Per dur a terme la pràctica de simulació basada en agents vam voler provar Mesa, un framework de Python que es va fer públic el 2015, amb l'objectiu desenvolupar tasques de simulació de forma fàcil, visual i amb la possibilitat d'analitzar les dades utilitzant les eines que Python disposa. Per a poder-lo utilitzar, vam tenir certs problemes els quals els vam solucionar tal com està explicat en l'apartat Annex de la pàgina 9.

Creiem que aquest entorn de treball es força similar als entorns presentats Netlogo i Repast. S'ha de dir que al principi ens ha sigut complicat trobar informació i exemples per veure com funcionava, però un cop trobada, l'enteniment del funcionament de l'entorn ha sigut més fàcil. En general creiem que ha sigut una bona experiència treballar amb Mesa, però veiem que encara hi ha poca informació d'aquest framework, ja que és bastant nou.

Sintaxis per a definir els agents en Mesa

Per poder definir un agent en el model de mesa s'ha de crear una classe dins del fitxer agents.py, en el qual anomenarem el nom de l'agent, les variables que aquest disposarà i com actuarà. A més a més, aquest heretarà totes les funcions de la classe específica que li passem per paràmetre. En la figura 1, es pot veure implementat un dels nostres agents dins del nostre model.

```python
class Egg(RandomWalker):

    energy = None

    def __init__(self, unique_id, pos, model, moore, energy=None, fecundat=False, current_step=None):
        super().__init__(unique_id, pos, model, moore=moore)
        self.energy = energy
        self.pos = pos
        self.fecundat = fecundat
        self.current_step = current_step

    def step(self):

        """
        Egg has "energy" and decreases if it is not fertilized
        """

        self.current_step += 1
        self.energy -= 1

        if self.energy <= 0:
            # Egg dies (0 energy)
            self.model.grid._remove_agent(self.pos, self)
            self.model.schedule.remove(self)

        else:
            # Egg is alive
            decimal = abs(self.current_step/12) - abs(int(self.current_step/12))
            comparar = abs((9-1)/12) - abs(int((9-1)/12))

            if not(self.current_step%12==0 or round(decimal,2)>round(comparar,2)):
                this_cell = self.model.grid.get_cell_list_contents([self.pos])
                cuantos_gallos_encima = [obj for obj in this_cell if isinstance(obj, Rooster)]
                if(len(cuantos_gallos_encima)>0):
                    # Egg changes to Fertilized Egg (Rooster at the same cell)
                    fertilizedEgg = FertilizedEgg(
                            self.model.next_id(), self.pos, self.model, self.moore, 0, self.fecundat, self.current_step
                    )
                    self.model.grid.place_agent(fertilizedEgg, self.pos)
                    self.model.schedule.add(fertilizedEgg)
                    self.model.grid._remove_agent(self.pos, self)
                    self.model.schedule.remove(self)
```

Fig. 1: Implentació del agent "Egg"el nostre model

Per tal de poder disposar finalment d'aquest agent i que sigui visible, s'ha d'afegir dins de l'arxiu server.py el nom d'aquest agent com podem veure en la figura 2. A més a més, s'ha de guardar una imatge dins de la carpeta de recursos, la qual s'utilitzarà per distingir el nostre agent dels altres quan l'estiguem simulant.

```python
if type(agent) is Rooster:
    portrayal["Shape"] = "wolf_sheep/resources/rooster.png"
    portrayal["scale"] = 0.9
    portrayal["Layer"] = 2
    portrayal["text"] = round(agent.energy, 1)
    portrayal["text_color"] = "White"
```

Fig. 2: Definició del aspecte visual del notre agent

Definir l'entorn on conviuen els agents

Les variables del model es poden modificar dins de l'arxiu server.py, on podem declarar quins agents volem analitzar en el gràfic i amb quin color mostrar-ho.

```
canvas_element = CanvasGrid(wolf_sheep_portrayal, 20, 20, 500, 500)
chart_element = ChartModule(
    [{"Label": "Hen", "Color": "blue"}, {"Label": "Rooster", "Color": "red"},{"Label": "HatchingChicken", "Color": "yellow"}
    ,{"Label": "Egg", "Color": "orange"}, {"Label": "FertilizedEgg", "Color": "green"}]
)
```

Fig. 3: Definició dels agents a observar

També dins d'aquest mateix fitxer podem declarar quines variables podrem modificar el valor, per tal de veure diferents casos dins del model.

```
model_params = {
    "grass": UserSettableParameter("checkbox", "Grass Enabled", True),
    "grass_regrowth_time": UserSettableParameter(
        "slider", "Grass Regrowth Time", 20, 0, 50
    ),
    "initial_rooster": UserSettableParameter(
        "slider", "Initial Rooster Population", 1, 1, 100
    ),
    "initial_hen": UserSettableParameter(
        "slider", "Initial Hen Population", 1, 1, 100
    ),
    "initial_hatchingChicken": UserSettableParameter(
        "slider", "Initial Hatching Chicken Population", 1, 1, 100
    ),
    "initial_egg": UserSettableParameter(
        "slider", "Initial Egg Population", 1, 1, 100
    ),
    "initial_fertilized_egg": UserSettableParameter(
        "slider", "Initial Fertilized Egg Population", 1, 1, 100
    ),
    "rooster_gain_from_food": UserSettableParameter(
        "slider", "Rooster Gain From Food", 4, 1, 10
    ),
    "hen_gain_from_food": UserSettableParameter(
        "slider", "Hen Gain From Food", 4, 1, 10
    ),
    "hatchingChicken_gain_from_food": UserSettableParameter(
        "slider", "Hatching Chicken Gain From Food", 4, 1, 10
    ),
}
```

Fig. 4: Definició dels parametres per controlar el model

Funcions de suport per a la modelització del comportament dels agents

Per facilitar el procés de programació hem utilitzat dues llibreries de python, una que és la de random i l'altre que és la de math;
La primera d'elles, l'hem utilitzat en els casos que un dels animals volgués anar a una posició de gespa, aleshores hem guardat totes les possibles posicions en una llista, ja que no volíem anar sempre a la primera de les possibilitats. Gràcies a tenir emmagatzemades totes les possibles solucions, hem pogut fer un random sobre aquestes, d'aquesta forma no teníem un camí predeterminat dels diferents agents.

```
# Rooster has less than 5 of energy and he can't "fertilize" an egg
gespa_posicions = []

# Rooster looks grass cells to eat
for i in range(len(lista_posiciones_cercanas)):
   this_cell = self.model.grid.get_cell_list_contents([lista_posiciones_cercanas[i]])
   grass_patch = [obj for obj in this_cell if isinstance(obj, GrassPatch)][0]

   if grass_patch.fully_grown:
       gespa_posicions.append(lista_posiciones_cercanas[i])

if(len(gespa_posicions)>0):
    # Rooster choses randomly one grass cell to go
    self.move_position(random.choice(gespa_posicions))
else :
    # Rooster choses randomly where to go (no grass cell around him)
    self.random_move()
```

Fig. 5: Exemple d'ús de la llibreria random

En segon lloc, també hem utilitzat la libreria math, amb l'objectiu de poder calcular la distancia entre dos punts de la manera més fàcil possible, sense haver de programar nosaltres elements extres. Gràcies a aquesta funció, hem pogut calcular la distància més pròxima de tants dels Galls, per tal de poder anar més ràpidament on s'havien post ous anteriorment i poder-los fertilitzar, com per les Gallines, per poder anar on havien posat i tornar-ho a fer. A continuació, podeu veure la funció que hem creat que utilitza math per ajudar-nos en tot aquest procés.

```
def distance(p0, p1):
    # Calculate distance between 2 points
    return math.sqrt((p0[0] - p1[0])**2 + (p0[1] - p1[1])**2)
```

Fig. 6: Exemple d'ús de la llibreria math

Modelització d'un sistema basat en agents

Nosaltres hem partit del model “wolf-sheep” que ens vam descarregar del repositori de GitHub de Mesa mitjançant aquest enllaç: https://github.com/projectmesa/mesa (en el qual hi ha exemples de diferents models)

Primerament, vam voler entendre el model dels llops i ovelles que ja estava creat amb el fi de poder fer-ne un encara més complex. En canvi d’utilitzar llops i ovelles, nosaltres hem volgut crear galls, gallines, pollets, ous i ous fertilitzats per tal de donar més complexitat, tant per la necessitat de controlar més agents com pel fet que es moguessin amb més intel·ligència, sense haver de moure's a l’atzar; prenent les millors decisions per la seva subsistència tant de moviment com de comunicació amb altres agents.

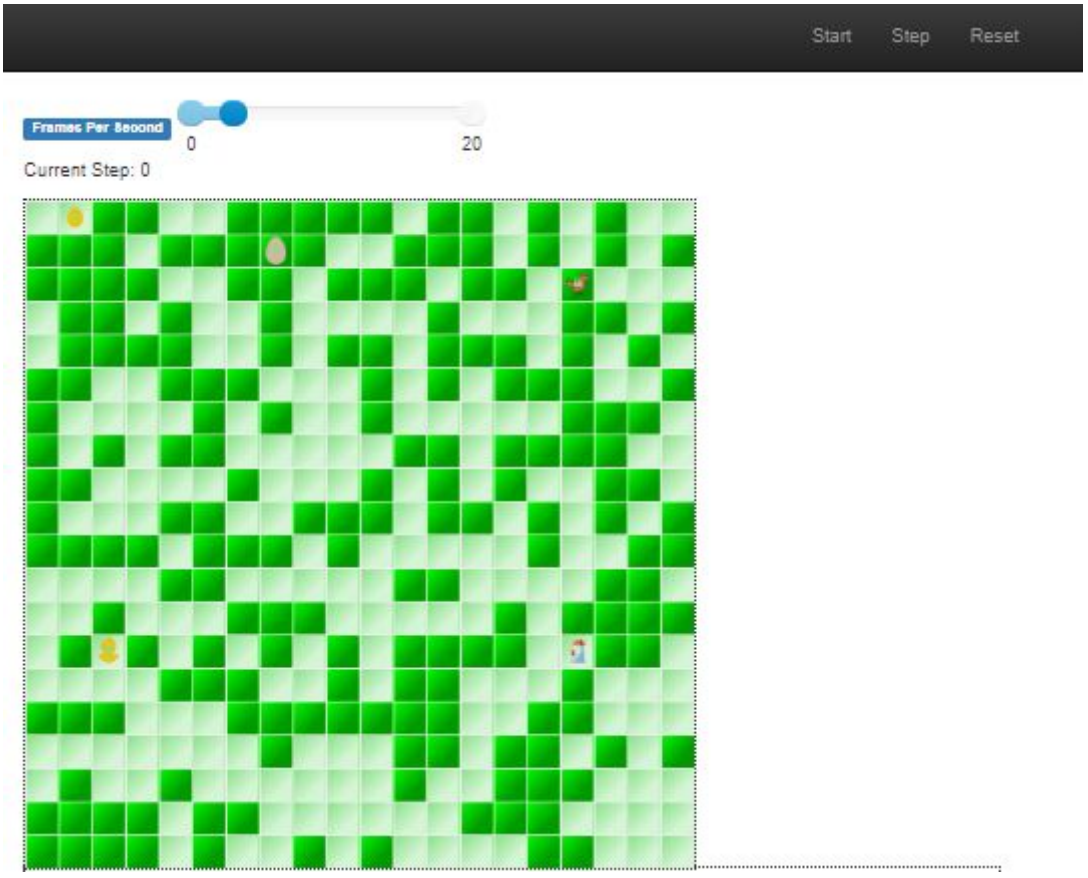

Fig. 7: Execució del model

Hem representat un dia en dotze steps, per tal de poder veure el cicle de les gallines i els galls en un corral. Tots els animals del step 8 al 12, ambdós inclosos dormen, mentre que en altres estan en moviment tant per trobar menjar com per reproduir-se. Els galls principalment sempre que tenen energia suficient, en el nostre cas hem decidit que cinc seria el mínim, buscaran els ous de les gallines per fecundar-los. Mentre que en el cas que no tinguin suficient energia, aquest estaran buscant gespa per poder menjar i recuperar energia. En el moment que els galls trobin un ou de gallina se'l memoritza per saber on normalment una gallina pon ous, per tornar en fases posteriors. També en el cas que trobi una gallina, aquesta li comunicarà on ha post ous, d'aquesta manera els galls tindran la informació per poder anar cap a aquelles direccions on és més probable trobar un ou per fecundar.

Les gallines depenent del step on es troben, actuaran d'una forma o d'un altre, ja que hem buscat que la gallina aproximadament pon un ou per dia. Per tant, del step 1 al 4, la gallina intentarà anar a pondre ous on ja els havia posat, en el cas que no hagués posat cap anteriorment, els posarà on estigui. A més a més, durant aquest període les gallines intentaran tornar cap on havien posat abans, ja que durant les steps de 5 al 8, aquestes gallines s'allunyaran en cerca de gespa per alimentar-se i poder pondre ous més endavant. Cal afegir que les gallines tenen memòria com els galls, ja que sempre se'n recordaran on han post ous, per tal de potenciar la subsistència de l'espècie. Tant el gall com la gallina a l'hora d'anar a pondre ous o anar a buscar-los per fecundar, calcularan quina distància és la menor i quin és el millor moviment per arribar abans.

Quan un ou hagi sigut posat per la gallina, aquest tindrà una certa vida i anirà disminuint a la mesura que no hagi estat fecundat. Per tant, l'objectiu dels galls i gallines és intentar fer que la reproducció sigui el més ràpid possible, les gallines intentaran comunicar als galls on trobar els ous (recordant-se d'on els havien post ous), i els galls escolliran quin ou fecundar en funció de quin tingui menys energia , per evitar que l'ou mori. A més a més, quan aquests ous hagin sigut fecundats, es tornen grocs en el nostre model, per poder veure més fàcilment la diferència entre ou fecundat i no fecundat. Aquests ous fecundats tindran d'esperar un cert temps abans d'aparèixer el pollet, que aquest mateix al cap d'un cert temps tindrà les mateixes possibilitats de passar a ser gall o gallina. Per tant, com podem veure el nostre model no es basa en agents que es moguin sense sentit, sino que intenten perpetuar la continuïtat de la seva espècie pel seu bé. Per últim, cal dir que la gespa s'anirà reproduint al cap d'un cert temps, perquè els diferents animals puguin alimentar-se amb continuïtat.

També es poden modificar el nombre d'animals de cada tipus com l'energia que guanyen a l'hora de menjar.

Fig. 8: Parametres per configurar el model

Visualment durant tots els steps del model, podrem veure l'evolució de la població de cada agent en una gràfica, en la imatge següent es pot veure un possible cas.

Fig. 9: Gràfiques de població dels agents del model

Annex

Per a instal·lar Mesa, en el nostre cas a Windows 10, hem trobat alguns problemes durant el procés que vam haver de resoldre per tal de poder utilitzar l'entorn. Tot seguit documentarem els errors trobats i com els hem pogut solucionar.

Per tal de poder instal·lar Mesa, és necessari tenir instal·lat Python prèviament, el qual es pot aconseguir anant a la web oficial: (https://www.python.org/downloads/, en el nostre cas ens vam descarregar l'última versió, Python 3.8.3. Tot seguit, és necessari crear una variable d'entorn per al bon funcionament de Python (afegir el PATH a les variables d'entorn de l'ordinador).

Un cop tinguem instal·lat Python, i comprovem que ens funciona bé, ens caldrà instal·lar "pip", el qual ens permetrà instal·lar "pipenv" (que és l'eina necessària per instal·lar Mesa). Per tal d'aconseguir pip, vam trobar una pàgina web que especificava de forma bastant clara els passos a seguir, aquesta pàgina web és la següent: https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/. En aquesta pàgina s'explica que cal guardar un arxiu de forma local i després executar la comanda "python get-pip.py". Això ens permetrà instal·lar-nos "pip".

Per instal·lar “pipenv” només és necessari executar la comanda #pip install pipenv. Un cop instal·lat “pipenv”, vam procedir a instal·lar Mesa amb la comanda “pipenv install mesa”.

Per tal de solucionar les incompatibilitats entre la nova versió de Python i Mesa només cal afegir les dues primeres línies que podeu veure en la imatge, dins de run.py, per tal de solucionar el problema d’incompatibilitat, i poder utilitzar Mesa.

```
import asyncio
asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
from wolf_sheep.server import server

server.launch()
```

Fig. 10: línies per fer funcionar mesa

Per últim per poder executar Mesa cal posar la comanda “mesa runserver PATH TO FOLDER WITH PROJECT”