PAC

Presentació

Tercera activitat d'avaluació continuada del curs. En aquesta PAC es pretén conèixer i desenvolupar sistemes multiagent.

Competències

Competències de grau
- Capacitat per utilitzar els fonaments matemàtics, estadístics i físics i comprendre els sistemes TIC.
- Capacitat per analitzar un problema en el nivell d'abstracció adequat a cada situació i aplicar les habilitats i coneixements adquirits per abordar-lo i resoldre'l.
- Capacitat per conèixer les tecnologies de comunicacions actuals i emergents i saber-les aplicar, convenientment, per dissenyar i desenvolupar solucions basades en sistemes i tecnologies de la informació
- Capacitat per proposar i avaluar diferents alternatives tecnològiques i resoldre un problema concret

Competències específiques
- Capacitat per utilitzar la tecnologia d'aprenentatge automàtic més adequada per a un determinat problema.
- Capacitat per avaluar el rendiment dels diferents algorismes de resolució de problemes mitjançant tècniques de validació creuada.

Objectius

L'objectiu d'aquesta PAC és conèixer el funcionament d'un entorn de desenvolupament de sistemes multi-agent. En concret es treballarà amb l'entorn Mesa de Python (https://github.com/projectmesa/mesa). Es proporciona la implementació d'un sistema multi-agent i es demana l'anàlisi i la implementació de canvis en el sistema.

Descripció de la PAC

Analitzeu els arxius dins de l'arxiu **Wolf_sheep.zip** Es tracta d'una simulació d'un model ecològic simple amb 3 agents: llops, ovelles i herba. Llops i ovelles gasten energia movent-se, mentre que la recarreguen menjant (el llop menja ovelles i aquestes herba). Si el llop / ovella té prou energia es reprodueix (per simplicitat únicament es necessita un agent), mentre que l'herba creix en cada iteració. La Figura 1 mostra l'execució de la simulació.

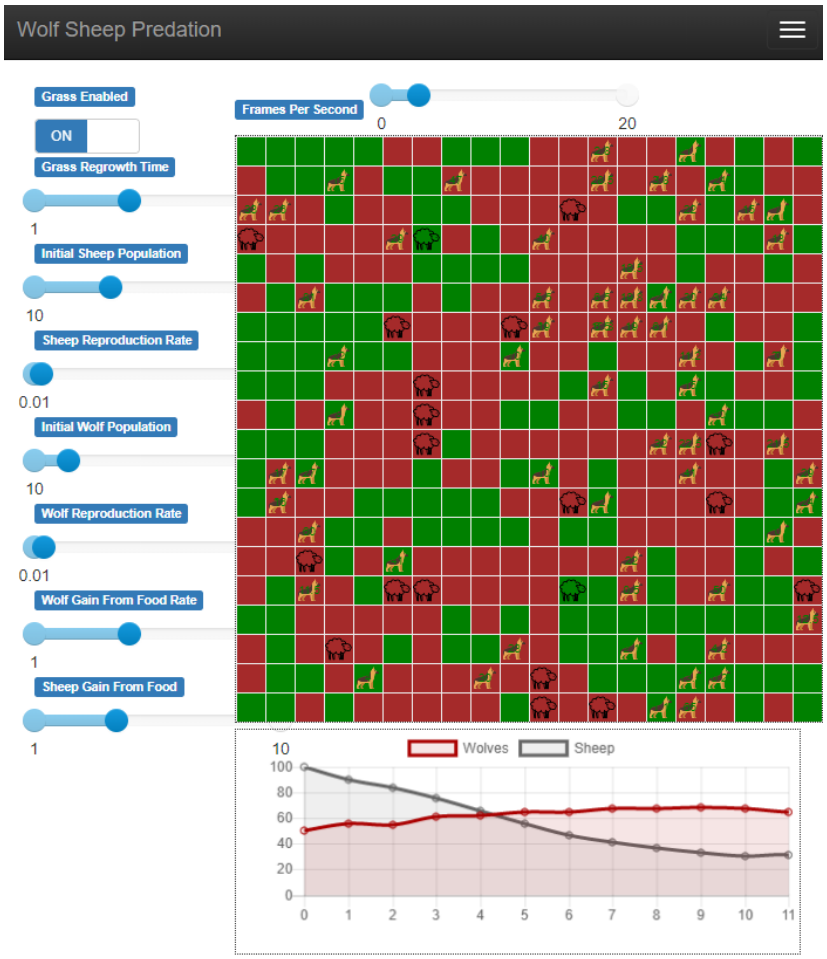


Figura 1. Simulació de l'exemple Wolf-Sheep.

En l'arxiu .zip podeu trobar els següents arxius:

- run.py (fitxer per executar la simulació en Python ($python run.py).
- agents.py: definició dels agents del model.
- Model.py: definició del model del sistema (nombre i tipus agents, paràmetres de simulació, etc.).
- Random_walk.py: definició genèrica d'un agent caminant (en què es basaran els agents ovella i llop).
- Schedule.py: gestiona els esdeveniments que succeeixen a cada pas de la simulació (què els passa als agents en cada iteració). Bàsicament controla el nombre d'agents i executa la funció step definida en cada agent (veure agents.py)
- Server.py: definició de la interfície gràfica per a la simulació, paràmetres de la simulació, valors a mostrar, gràfiques, etc (veure Figura 1).

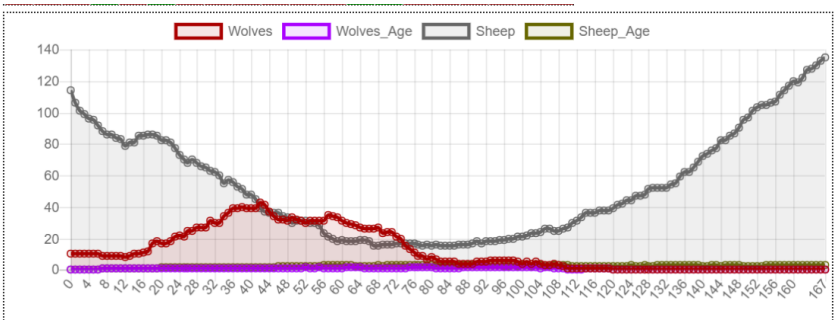
Exercicis:

1. Identifiqueu els agents que hi ha al sistema i definiu el seu tipus. Descriviu les variables i accions que pot desenvolupar cada agent.

   Tenim 3 agents en el sistema, tots de tipus reactiu i sense la capacitat de comunicar-se amb altres agents.
   - Herba: agent que únicament creix i és menjat per sheep.
     - Variables
       - Countdown: enter que compta el nombre de cicles que triga a créixer l'herba (establir fully_grown)
       - Fully_grown: booleà que controla si ha crescut l'herba.
     - Accions: dins la funció step, decrementar countdown fins arribar a 0 quan fully_grown passa a valer cert.
   - Sheep / Wolf: agent que es desplaça de manera aleatòria pel model menjant agents herba / sheep si estan en la seva mateixa posició, amb capacitat de reproducció (crear nous agents).
     - Variables
       - Energy: energia de l'agent.
       - x, y: posició en el model de simulació (cel·la).
   - Accions: dins d'step, en cada iteració es decrementa la seva energia i l'agent es mou de manera aleatòria (acció random_move). Si els agents mengen herba / sheep (hi ha un agent d'aquest tipus a la seva cel·la) augmenten la seva energia. En el cas d'herba s'incrementa countdown i en el cas de sheep es mata l'agent sheep (veure remove_agent i acció remove). Si l'agent es queda sense energia l'agent es mor i si aquest ha de reproduir-segons la probabilitat de reproducció (paràmetre del sistema) divideix la seva energia per 2 i crea un nou agent del mateix tipus en la mateixa posició.

2. Executeu una simulació del sistema analitzar l'impacte dels paràmetres del model en les poblacions dels agents. Modifiqueu els agents sheep i Wolf afegint la variable edat que s'incrementarà en cada iteració (ex. cada iteració / step pot equivaler a 1 mes). Els agents podran reproduir-se si tenen més d'1 any i poden arribar a una edat màxima de 20 anys. Executeu la nova simulació obtenint la gràfica de l'edat mitjana de cada tipus d'agent. (Ajuda: veure DataCollector a l'arxiu model i la funció get_breed_count ()).

   Segons els paràmetres per defecte del sistema el depredador (wolf) té un impacte sobre la població de la presa (sheep), de manera que la població de sheep disminueix molt ràpidament, i conseqüentment, després d'unes iteracions ho fa també la població del depredador . En el cas que el nombre de depredadors sigui menor que el de les preses, el nombre de presa augmentaria de manera descontrolada (sent aquesta una espècie invasora), tal com mostra la següent figura, on la població sheep ha baixat tant que la població de Wolf ha disminuït fins a desaparèixer. Sense depredador, la població de sheep augmenta de manera gradual. Aquest sistema és una simplificació del modelatge de presa-depredador, també conegut com equacions Lotka-Volterra (https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations).



Referent a les modificacions del sistema, s'han realitzat els següents canvis:

- Random_walker: s'han afegit els atributs age i ageMax (ja que serà igual para Wolf i sheep). S'ha afegit la funció step per incrementar l'edat

```
def step(self):
        self.age = float(self.age) + float(1.0/12.0)
```

- Agent.py: s'ha modificat la funció de mort de l'agent depenent ara també en l'edat màxima. També s'ha modificat l'opció de reproducció afegint l'edat a partir d'1 any. El codi següent és per a l'agent Sheep però seria similar per al Wolf.

```
# Death
if self.energy < 0 or self.age > self.ageMax:
self.model.grid._remove_agent(self.pos, self)
        self.model.schedule.remove(self)
        living = False

if  living  and  self.age  >=1  and    self.age  <  self.ageMax  and
random.random() < self.model.sheep_reproduce:
      # Create a new sheep:
      if self.model.grass:
            self.energy /= 2
            lamb    =    Sheep(self.pos,    self.model,    self.moore,
self.energy)
            self.model.grid.place_agent(lamb, self.pos)
            self.model.schedule.add(lamb))
```

- En server s'ha afegit nova informació per mostrar en les gràfiques, en ChartModule:

```
        {"Label": "Wolves", "Color": "#AA0000"},
        {"Label": "Wolves_Age", "Color": "#AA00FF"},
        {"Label": "Sheep", "Color": "#666666"},
        {"Label": "Sheep_Age", "Color": "#666600"}]
```

- En scheduler, s'ha afegit la funció que retornar l'edat mitja d'un tipus d'agent (s'executarà aquesta funció en datacollector del fitxer model.py)

```
def get_breed_meanAge(self, breed_class):
                '''
        Returns the current number of agents of certain breed in the
queue.
        '''
        numAgents = len(self.agents_by_breed[breed_class])
        ageCount = 0.0
        meanAge = 0.0
        for agent_class in self.agents_by_breed[breed_class]:
            ageCount = ageCount + agent_class.age
        if numAgents > 0:
            meanAge =  float (ageCount) / float (numAgents)
        return meanAge
```

- En model.py, el datacollector (que dóna informació al scheduler sobre els agents del sistema) s'ha actualitzat per donar la informació de l'edat.

```
self.datacollector = DataCollector(
   {"Wolves": lambda m: m.schedule.get_breed_count(Wolf),
    "Wolves_Age": lambda m: m.schedule.get_breed_meanAge(Wolf),
    "Sheep": lambda m: m.schedule.get_breed_count(Sheep),
    "Sheep_Age": lambda m: m.schedule.get_breed_meanAge(Sheep)}
              )
```

Amb aquestes modificacions, la població de wolf disminueix més ràpidament (no tenia depredador), sent un sistema una mica més realista. S'observa que l'edat mitjana de sheep és sempre molt inferior a wolf ja que té sempre un depredador.

3. En un sistema multi-agent on els agents sheep i Wolf poden comunicar-se entre ells, penseu 2 tipus d'accions podrien realitzar per fer més realista el sistema, quin protocol FIPA utilitzarien i que informació es passarien.

Els agents del mateix tipus es podrien comunicar per passar-se informació sobre la posició del depredador / presa per ser més eficients. En el cas de sheep organitzar-se per agrupar-se i sobrepassar en nombre l'agent Wolf per evitar ser menjats. En aquest cas podrien utilitzar un FIPA-Request amb la informació de la posició del depredador i la posició per a realitzar l'agrupament. De manera similar Wolf podrien organitzar-se per acorralar agents sheep, mitjançant un FIPA-ContractNet un agent Wolf presentaria les condicions: informació de la posició de la presa, energia de la presa i altres possibles condicions (temps estimat de la caça, energia que es gastarà, i premi en el cas que la caça sigui un èxit). Si un agent Wolf considera que les condicions són favorables contestaria amb accept-proposal en cas contrari reject-proposal.

Recursos

Bàsics
Per realitzar aquesta PAC disposeu dels arxius en el zip wolf_sheep.zip el tutorial http://mesa.readthedocs.io/en/latest/tutorials/intro_tutorial.html, i la biblioteca d'exemples del Mesa (https://github.com/projectmesa/mesa/tree/master/examples/) així com els apunts del mòdul de sistemes multi-agent.

Criteris de valoració

Els tres exercicis d'aquesta PAC es valoraran amb 4, 3, 3 punts respectivament.
Raoneu la resposta en tots els exercicis. Les respostes sense justificació no rebran puntuació.

Format i data de lliurament

Cal lliurar la PAC en un fitxer zip amb el pdf de la memòria al registre d'activitats d'avaluació continuada.

El nom del fitxer ha de ser CognomsNom_AC_PAC3 amb l'extensió . zip (ZIP).

Data Límit: 25 de Maig de 2018 a les 24 hores.

Per a dubtes i aclariments sobre l'enunciat, adreceu-vos al consultor responsable de la vostra aula.

Nota: Propietat intel·lectual
Sovint és inevitable, en produir una obra multimèdia, fer ús de recursos creats per terceres persones. És per tant comprensible fer-ho en el marc d'una pràctica dels estudis d'Enginyeria Informàtica, sempre i això es documenti clarament i no suposi plagi en la pràctica.
Per tant, en presentar una pràctica que faci ús de recursos aliens, s'ha de presentar juntament amb ella un document en què es detallin tots ells, especificant el nom de cada recurs, el seu autor, el lloc on es va obtenir i el seu estatus legal: si l'obra està protegida pel copyright o s'acull a alguna altra llicència d'ús (Creative Commons, llicència GNU, GPL ...). L'estudiant haurà d'assegurar-se que la llicència que sigui no impedeix específicament seu ús en el marc de la pràctica. En cas de no trobar la informació corresponent haurà d'assumir que l'obra està protegida pel copyright.
Hauran, a més, adjuntar els fitxers originals quan les obres utilitzades siguin digitals, i el seu codi font si correspon.