PRÀCTICA ABM:

Espai natural

SIM

Laia Ondoño laia.ondono@est.fib.upc.edu
Anna Llanza anna.llanza@est.fib.upc.edu

QP 2019-2020 15/06/2020

Estudi del framework

Per tal de trobar un framework adequat per a realitzar la simulació d'un model orientat a agents que s'ajustés a les nostres necessitats i als nostres coneixements, vam decidir realitzar una cerca per Internet fixant-nos sobretot en el llenguatge de programació, en si era de codi obert i en si era un framework actualitzat.

Primerament, vam trobar SPARK, un framework que utilitza Java però amb el qual vam tenir problemes de compatibilitat entre les versions d'aquest i del nostre ordinador. Després vam trobar MASON, que també era en Java però vam tenir problemes a l'hora d'executar la seva interfície perquè no aconseguia carregar el model. Finalment, ens vam decidir per utilitzar el framework Mesa, ja que vam veure que la seva última versió era bastant actual, disposava d'un tutorial amb una explicació pas a pas per a endinsar-nos en el seu funcionament i no vam tenir problemes per a executar. Aquest manual ens va ajudar a veure com generar un model, generar els seus agents, definir el seu comportament a cada tick, generar gràfiques resultants i visualitzar l'execució del model. També incloïa una carpeta amb exemples, dels quals vam poder veure com es podien modificar paràmetres d'entrada pel model i pels agents. A més, un altre punt a favor per a escollir aquest framework va ser que, quan buscàvem informació d'aquest framework, vam veure que tenen com a objectiu ser equivalents a altres frameworks com ara NetLogo, Repast o MASON, dels quals, a classe vam poder veure una petita part del seu funcionament i vam creure que ens seria més fàcil adaptar-nos a aquest framework.

Mesa es un framework que utilitza únicament Python que es centra en la modelització basada en agents. Permet generar models amb components que ja venen creades, com ara graelles, a més d'afegir una interfície al navegador amb la qual visualitzar l'execució del model. També incorpora l'opció de generar gràfiques i taules per a veure l'evolució durant l'execució d'alguns paràmetres del model i/o dels agents.

Mesa té 3 components: el component de modelatge, el d'anàlisis i el de visualització. Aquest components estan separats, però estan pensats per a ser utilitzats conjuntament. El component de modelatge implementa les classes mesa.Model, mesa.Agent, amb les quals hem generat el nostre model i els seus agents respectivament, a més de la classe mesa.time, utilitzada per a planificar l'ordre i execució dels agents a cada step, i el mesa.space per a generar la graella en què els agents es mouen. Del component d'anàlisis només hem utilitzat el mesa.datacollection que ens ha permès recollir dades de l'execució del model per a generar gràfiques. Finalment, els mòduls de visualització que hem utilitzat han estat el mesa.visualization.ModularVisualization, per a poder executar el model en un servidor web, i el mesa.visualization.modules per a generar el contingut d'aquesta pàgina del navegador en la qual s'ha executat el model.

Per tal de poder entendre millor el model, llistem una sèrie de classes, mètodes i atributs que hem utilitzat per a la realització d'aquesta pràctica.

Classe Model:

- self.grid = MultiGrid(width, height, True) : es genera la graella on s'executarà el model i s'assigna a l'atribut grid.
- self.schedule = RandomActivation(self) : configura que l'ordre d'execució dels agents a cada step sigui aleatori i s'assigna a l'atribut schedule.
- self.schedule.add(agent) : afegeix l'agent a l'atribut schedule, aquest atribut contindrà tots els agents que s'executaran a cada step.
- self.grid.place_agent(agent, (posx, posy)) : col·loca l'agent a la posició indicada de la graella.
- def step(self) : aquest mètode s'executarà a cada step de l'execució del model.

Classe Agent:

- super().__init__(id, model) : genera una instància de la classe Agent de Mesa identificada per l'enter id i vinculada al model model.
- def step(self) : aquest mètode s'executa a cada tick del rellotge.
- possible_steps = self.model.grid.get_neighborhood(self.pos, moore=True, include_center=False) : guarda a possible_steps les cel·les a les qual l'agent es pot moure. Aquesta decisió pot seguir la funció de transició de Moore o de Von Neumann i pot incloure com a opció la posició de la cel·la actual.
- cellmates = self.model.grid.get_cell_list_contents([cel·la]) : aquest mètode retorna un vector amb els agents que es troben en aquella cel·la de la graella.
- self.model.grid.move_agent(self, nova_posicio) : aquesta funció desplaça l'agent a la nova posició.

Model

En aquesta pràctica hem volgut modelar una espai natural on hi ha boscos de bolets. Aquests boscos poden ser de bolets verinosos. Diàriament hi ha persones que van a collir bolets per tal d'alimentar-se però, al ser persones que no tenen gaire coneixement sobre les diverses espècies dels bolets d'aquest espai natural, poden menjar-se bolets verinosos i, conseqüentment, enverinar-se. Una persona només aprendrà de l'existència de bolets verinosos si, malauradament, en menja. A més, aquest espai natural disposa d'un curador que és l'encarregat de curar els bolets per tal de fer-los no verinosos i així evitar que els visitants s'enverinin.

Per a construir el nostre **model**, hem utilitzat ModularServer, on hem instanciat el model, passant-li els paràmetres d'entrada, i hem configurat i generat la graella on es veurà l'execució i la gràfica on es veurà l'evolució d'una de les variables d'un agent. En instanciar el model, es generen els agents que aquest contindrà i es col·loquen en una posició inicial de la graella.

El mètode amb el qual s'inicialitza una instància de la classe Model comença creant la graella i el planificador. Després genera els agents en funció de la quantitat indicada als sliders del navegador i els col·loca en una posició aleatòria de la graella. El model s'ha d'assegurar que en una mateixa posició no hi ha 2 o més agents de tipus BoscBolets. Finalment genera el recolector de dades, assignant el mètode que s'executarà per a obtenir-les a cada step. El mètode step comença recollint les dades i guardant-les al datacollector i després fa una crida a que s'executi el planificador (atribut schedule). Com hem dit anteriorment, aquest planificador conté tots els agents del model, planifica l'ordre en què aquests s'executaràn i crida a la seva execució.

L'agent **AgentBoscBolets** està programat per a ser un agent simple. Aquest té com a atributs principals la quantitat de bolets que té el bosc, un booleà indicant si els seus bolets són verinosos o no. A la creació d'una instància d'aquest agent, cal passar com a paràmetres la quantitat de bolets del bosc i si aquest bosc és verinós o no. Al ser un agent simple, aquest només serveix per a alimentar a un AgentPersona. És per això, que al mètode step només mira si està mort i, en cas afirmatiu, comença un compte enrere en el qual simulem que el sòl s'està regenerant. Independentment de si abans era verinós o no, quan el bosc mor es torna verinós. Una vegada s'hagi acabat aquest compte enrere, el bosc es començarà a regenerar, fent que a cada step creixin 10 bolets. Aquest creixement s'aturarà una vegada el bosc tingui la quantitat de bolets inicial.

L'agent **AgentPersona** és un agent basat en model perquè té memòria del que ha fet. Com a atributs inicials, té la vida, un booleà que indica si està enverinada al menjar un bolet o no i un vector en el qual registrarà els boscos de bolets verinosos que ha menjat, per tal de no tornar-se'ls a menjar. El valor que tindrà l'atribut vida s'assignarà a la inicialització de la instància amb el paràmetre rebut.

A cada step, el seu moviment està condicionat per les seves condicions actuals. Si la persona té poca vida o està enverinada, es mourà a un bosc adjacent, si n'hi ha, per tal de

menjar bolets i sobreviure, si li falta poca vida, o curar-se, si està enverinada (si una persona està enverinada i menja bolets no enverinat es cura). Per a escollir a quin bosc adjacent es mou, mirarà a la seva memòria on registra els boscos verinosos dels quals ha menjat bolets, per tal d'evitar-los. En el cas de que la persona estigui sana, es mourà a una posició aleatòria. Una vegada la persona s'ha mogut, mira si està a una casella on hi ha un bosc de bolets. Si no és així, la persona no fa res. En cas contrari, es menjarà una quantitat de 10 bolets del bosc (decrementem l'atribut quantitat del bosc corresponent). Una vegada se'ls ha menjat, veurà si són bons o verinosos. En cas de que siguin verinosos, la persona passarà a estar enverinada, se li restarà vida i afegirà a la seva memòria el coneixement de que a aquesta posició hi ha un bosc verinós. Si el bosc no es verinós, se li sumarà vida i, si la persona estava enverinada, es curarà.

Finalment tenim l'agent **AgentCurador**, un agent basat en objectius, donat que abans de prendre qualsevol decisió, avalua i revisa el conjunt d'accions que el poden apropar el seu objectiu. L'objectiu del curador és curar el màxim número de boscos de bolets verinosos possible. Aquest agent, té com a atributs principals l'antidot, el qual serveix per a curar els boscos de bolets i un vector dels boscos curats en l'últim trajecte. En la creació, se li donarà valor a l'antidot a partir del valor del slider del navegador. El step està condicionat per si l'agent està recarregant l'antídot o no. Si no l'està recarregant, mirarà si té boscos verinosos adjacents. Per a fer això, mirarà a totes del cel·les que té al voltant i afegirà en un vector els boscos que estiguin en aquestes cel·les, ordenats per la quantitat de cada bosc. Si el vector està buit, es mourà a una posició aleatòria. Si no ho està, l'agent haurà de mirar si li surt a compte o no curar algun dels boscos de bolets que s'ha trobat.

Per tal d'avaluar quina és la millor acció a realitzar, l'agent mirarà cada bosc i escollirà el que ell consideri que té una millor repercussió per a aconseguir el seu objectiu. Per a prendre aquesta decisió, l'agent considerarà els boscos que tenen una quantitat inferior o igual a 30 com a boscos que no són prioritaris per a ser curats (interès 0). Si la quantitat es troba entre 30 i 150, l'agent mirarà si té suficient antídot com per a curar 4 boscos d'aquestes dimensions. Si és així, l'interès que l'agent tindrà en aquest bosc serà de 2. Si això no es compleix però té suficient antídot per a curar el bosc, assignarà un interès de 1. Finalment, si la quantitat del bosc és superior a 150, mirarà si té suficient antídot com per a curar dos boscos d'aquestes dimensions. Si és així, assignarà l'interès més alt (interès 4) i, en cas contrari, interès de 3. D'aquesta manera, el curador prendrà la decisió buscant el bé major, és a dir, buscant curar el major nombre de bolets. Si no li interessa cap bosc i s'està esgotant l'antídot, l'agent curador anirà a recarregar-lo. Si ha decidit que no li interessa cap bosc es mourà a una posició aleatòria, evitant els boscos que té al costat. Si troba un bosc que li interessa, es mourà a la posició on el bosc es trobi.

Si ha decidit curar un bosc, se li restarà la quantitat de bolets que ha curat al seu antídot, i el bosc es tornarà no verinós. El curador afegirà la posició del bosc curat a la seva llista de boscos curats perquè, si el curador es creua amb una persona, li informa d'aquests boscos que eren verinosos però que ja no ho són perquè els ha curat. Per a avisar-les, mirarà si té persones al seu voltant i, en cas afirmatiu, les informarà dels boscos que ha curat, fent que aquestes eliminin de la seva llista de boscos verinosos els boscos curats. Si el curador porta 20 steps buscant boscos però no en troba o trobant-ne sense tenir interès, anirà a

recarregar l'antídot, ja que si no té interés en curar boscos vol dir que s'està esgotant el seu antídot. En cas que el curador se li acabi l'antídot, s'anirà a recarregar-lo i buidarà la llista de boscos curats. Per tal de que el curador recarregui l'antídot hauràn de passar 10 steps per a omplir-lo al complet.

Per tal de facilitar la visualització de cadascun dels agents, hem establert una llista de colors i mides on cada color correspon a un agent que es troba en cert estat. A continuació us llistem les possibilitats:

AgentBoscBolets (cercle de radi 0.5):
- marró: un bosc de bolets no és verinós
- vermell: un bosc de bolets és verinós
- negre: un bosc de bolets està mort i s'està regenerant

AgentPersona (cercle de radi 0.7):
- gris: una persona que no està enverinada
- lila: una persona que està enverinada
- mida més petita que la inicial: a la persona li queda poca vida

AgentCurador (cercle de radi 0.4):
- groc: el curador està treballant

Per acabar, quan una persona estigui morta o el curador estigui recarregant l'antídot, el seu cercle estarà de color blanc, per a simular que no hi són.

Instruccions d'ús

Per a poder executar el nostre model, caldrà instal·lar-se una sèrie de dependències llistades a continuació:

- Python 3 : per a comprovar la versió de Python que es té instal·lada, només cal obrir una Terminal i executar la comanda:
  >>> python --version
- pip : a continuació us deixem un link on explica com es pot instal·lar aquest paquet per a ordinadors amb sistema operatiu Linux, Windows i macOS. https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/
- Mesa: per a descarregar aquest framework, només cal fer:
  >>> pip install mesa
  >>> pip install -r
https://raw.githubusercontent.com/projectmesa/mesa/master/examples/boltzmann_wealth_model/requirements.txt
  seguint els passos indicats al link següent a l'apartat de Installation:
  https://mesa.readthedocs.io/en/master/tutorials/intro_tutorial.html#
  IMPORTANT: instal·lar Mesa dins la carpeta del projecte

No cal tenir una llicència per a poder utilitzar aquest framework

Per tal de posar el model en funcionament, només caldrà executar el fitxer main.py fent:
>>> python main.py

Comentaris finals

Estem molt contentes d'haver trobat aquest framework tan adequat per a la realització d'aquesta pràctica. Pensem que ens ha facilitat la implementació, ja que hem disposat de tutorials per a entendre i aprendre el seu funcionament. Abans de posar-nos a implementar el nostre model, vam seguir 2 tutorials que ofereixen a la pàgina web on t'expliquen pas a pas com construir un model basat en agents en aquest framework. A més, ens ha ajudat bastant el fet de que la sintaxi que Mesa utilitza sigui només Python, ja que és un llenguatge amb el que ja hem treballat i ens hem sentit còmodes. Una vegada vam acabar d'implementar el model, vam decidir que seria interessant ampliar-lo afegint la opció de modificar diversos paràmetres d'entrada. Per a poder fer això, vam mirar als exemples que Mesa proporciona. Un altre avantatge de Mesa ha estat que no ens ha calgut instal·lar altres paquets a part del propi framework amb llibreries que aquest incorpora i altres llibreries de Python.

Comparant Mesa amb els frameworks dels quals hem après a teoria, podem dir que, tot i que NetLogo i Repast ens han ajudat a fer-nos una idea de com són els frameworks de ABM, considerem que Mesa és més entenedor, especialment per a algú que coneix Python i mai ha treballat en aquest tipus de projecte.