Netlogo 워크숍

이남형[1]

[1]연세대학교 경영연구소

October 30, 2017

이번 강의의 목표

- 목록(행렬) 만들기
- 모형 test 하기와 wishlist 만들어 보기

쇼핑 모형과 ODD 프로토콜

- 목적
  - 과일과 야채 거래 시장을 Netlogo로 구현
- 독립체
  - 소비자와 판매자
- 확률 과정
  - 쇼핑 리스트의 아이템
  - 판매자가 보유하는 아이템, 아이템의 소매 가격
- 초기화
  - 소비자의 수, 구매를 위한 이동 속도, 구매 대안의 수
  - 판매자의 보유 아이템 수
- 결과
  - 시간 변화에 따라, 구매하지 못한 아이템의 평균 개수
  - 평균 구매 속도

ODD와 Pseudo Codes

- Pseudo Codes $\in$ ODD
  - ODD는 ABM을 이용한 연구의 전체 설계도
  - Pseudo Codes는 ABM을 위한 프로그래밍 가이드
    - ABM을 위한 프로그래밍은 반드시 Netlogo일 필요는 없음
    - C++, JAVA, Python, Swarm, $\cdots$

Pseudo Codes

- 판매를 위한 12개의 과일과 채소 리스트 설정
- 12개의 과일과 채소의 도매 가격 설정 $\rightarrow$ 1–100 중 랜덤
- 판매자
  - 9개, 위치는 활동 공간의 중앙, 붉은색 집
  - 각각의 가게는 12개의 과일과 채소 중 랜덤하게 보유
    - 몇 개를 보유할 지 슬라이더를 이용하여 통제
    - 소매가격은 도매 가격에 1–30% 를 랜덤하게 마크업
- 소비자
  - 몇 명의 소비자를 둘지는 슬라이더로 선택할 수 있음
  - 위치는 랜덤하게 결정, 노란색 사람

- 소비자 속성
  - 1–8 종의 랜덤 아이템이 포함된 쇼핑 리스트
  - 방문하지 않은 판매자 리스트: 초기값 9
  - 사용 금액: 초기값 0
  - 소비자의 쇼핑 리스트 평균 길이를 계산
- 쇼핑
  - 가장 싸게 파는 판매자를 스캔
  - 소비자는 선택한 판매자를 방문
    - 재방문은 없음
    - 구매 기록 저장
    - 지불 가격 합산
  - 쇼핑 리스트가 비면 소비자는 집으로 돌아감
- 결과를 보고하고 그래프를 그림

Setup: 전역 변수 설정

```
globals
 [
  fruit-and-veg
  mean-items
 ]


breed [ shoppers shopper ]
breed [ traders trader ]

shoppers-own [ shopping-list ]
traders-own [ stock ]
```

Setup: 상품 목록 설정

```
to setup
 clear-all

 set fruit-and-veg
  [
  "apples" "bananas" "oranges" "plums" "mangos"
  "grapes" "cabbage" "carrots" "potatos" "lettuce"
  "tomatoes" "beans"
  ]

 let xs [ -12 -9 -6 -3 0 3 6 9 12 ]

 reset-ticks
end
```

Setup: 판매자 생성

```
to setup
...
 let xs [ -12 -9 -6 -3 0 3 6 9 12 ]
 foreach xs [ ?1 ->
  create-traders 1
  [
   set shape "house"
   setxy ?1 0
   set color red
   set stock n-of n-items-stocked fruit-and-veg
  ]
 ]
...
end
```

Setup: 소비자 생성

```
    create-shoppers n-shoppers
    [
     set shape "person"
     setxy random-pxcor random-pycor
     set color yellow
     set shopping-list n-of (1 + random 8) fruit-and-veg
    ]

    set mean-items mean [ length shopping-list ]
    of shoppers

  reset-ticks
end
```

Setup: 시각적으로 확인하기

- `Interface` 탭
  - `Button` → `Button` → `setup`
  - `Button` → `Slider`
    - → n-items-stocked, 0 – 1 – 5
    - → n-shoppers, 0 – 1 – 20
- 셋업 확인
  - `setup` 버튼 누르기
  - `n-shoppers` 바꿔보기
    - ok?
  - stock 체크하기
    - → `monitor` → `stock`
    - 에러가 정상. 에러 내용을 확인해보자.
    - 어떻게 체크할 수 있을까?
    - `show ( word "has " stock )`

Go-procedure: 소비자가 판매자에게 가기

```
to go
ask shoppers with [ not empty? shopping-list ]
  [
    let stall one-of traders

    face stall

    while [ patch-here != [patch-here] of stall ]
       [forward 0.005 * walking-speed]
  ]

tick
end
```

```
[forward 0.005 * walking-speed]

let purchases filter [ ?1 -> member? ?1 [stock] of stall ]
  shopping-list

foreach purchases
 [ ?1 -> set shopping-list remove ?1 shopping-list ]
 if empty? shopping-list [ set ycor -16 ]
 ]

set mean-items mean [ length shopping-list ] of shoppers

if mean-items = 0 [ stop ]


tick
```

- `Interface` 탭
  - `Button` → `Slider`
    - → walking-speed, 0 – 0.1 – 0.5
  - `Button` → `monitor` →
    - `mean-items`
    - Mean number of items left to buy,

go

기본 모형 실행

- Interface 탭
  - Button → Button →
    - go
- 모형 구동 확인
  - n-items-stocked, n-shoppers, walking-speed 바꿔 가면서
  - 끝?
- 가격!

가격과 행위자

- 가격 변수
  - 판매자: 각 아이템별 가격
    - 도매 가격으로 구입한 후 마크업(랜덤)을 붙인 소매 가격
  - 소비자: 소비액
    - 소비액 기록
    - 예산은? → 추후 연구과제(wishlist)

Setup: 가격 변수 설정

```
globals
 [
...
  fruit-and-veg-prices
 ]

shoppers-own [ shopping-list spent ]
traders-own [ stock prices ]

to setup
...
 set fruit-and-veg-prices n-values
  (length fruit-and-veg) [ 1 + random 100 ]

 let xs [ -12 -9 -6 -3 0 3 6 9 12 ]
```

Setup: 상점별 마크업 가격 부여

```
foreach xs [ ?1 ->
  create-traders 1
  [
;     show ( word "has " stock )
    set prices [ ]
    let mark-up ( 1 + random 30 ) / 100
    foreach stock [ ??1 ->
     set prices lput
     ( ( 1 + mark-up) *
      ( item ( position ??1 fruit-and-veg )
       fruit-and-veg-prices )
     ) prices
    ]
  ]
```

Go: 소비액 계산

```
to go

ask shoppers with [ ...

 foreach purchases [ ?1 ->
  [
    set spent spent + item ( position ?1 [ stock ] of
           stall ) [ prices ] of stall
    set shopping-list remove ?1 shopping-list
  ]
```

결과값: 소비액 관찰

- Interface 탭
  - Button → plot →
    - Spending
    - `plot mean [ spent ] of shoppers`

가격이 반영된 모형 실행

- Spending 창 관찰
- 모형의 약점은?
  - mean number of items left to buy 관찰

판매자 방문과 선택

- 소비자가 최초에는 랜덤하게 판매자 방문 $\rightarrow$ 이후 판매자를 선택
- 만약 판매자를 모두 방문했는데 원하는 상품이 없다면 쇼핑 중지

Setup: 방문한 판매자는?

```
shoppers-own [ shopping-list spent not-yet-visited ]
...
to setup
...
create-shoppers n-shoppers
[
  ...
  set not-yet-visited traders
  set shopping-list ...
]
```

Go: 방문한 판매자 기억시키기

```
to go

ask shoppers with [ not empty? shopping-list ]
 [
;    let stall one-of traders
   let stall one-of not-yet-visited
   ...
   set not-yet-visited not-yet-visited with
       [ self != stall]
   let purchases filter [ ...
 ]
```

Go: 구매 중지 및 구매 못한 상품 기록하기

```
to go

ask shoppers with [ not empty? shopping-list ]
 [
   let stall one-of not-yet-visited
   if stall = nobody
    [
     show ( word "No one sells " shopping-list )
     set shopping-list []
     stop
    ]
  ...
```

판매자 방문과 중지 반영 모형 실행

- `Interface` 탭
  - go 버튼 → 마우스 오른쪽 버튼 클릭 `Edit` →
    - 박스 `forever` 체크
- 끝?
  - 책에서는 소비자가 일부 판매자만 방문하고, 자신이 구입하고자 하는 상품을 가장 싸게 파는 판매자로부터 구매하는 알고리듬을 구현
  - 자신의 wishlist를 늘려가 보자