Dining Philosophers در نت لوگو

مسئله غذاخوردن فیلسوف‌ها یک مورد کلاسیک در همزمانی فرآیندهای درحال اجرا می‌باشد که اکثر دانشجویان رشته کامپیوتر با این مسأله آشنایی دارند و این مسأله برای بسیاری از پروسه های در حال اجرا که به طور مشترک به منابع محدود دسترسی دارند مناسب است.

این مسأله بدین صورت می‌باشد که گروهی از فیلسوف‌ها(از حداقل ۲) دور یک میز دایره‌ای نشسته‌اند و هر فیلسوف یک بشقاب ماکارونی دارد و هرکدام ممکن است در یکی از حالت‌های زیر باشند :

۱) درحال فکرکردن

۲) گرسنگی

۳) در حال خوردن ماکارونی

در بین هر دو فیلسوف یک عدد چنگال وجود دارد که به طور دقیق‌تر می‌توان گفت به ازای هر فیلسوف یک عدد چنگال وجود دارد و ماکارونی‌ها به قدری لغزنده هستند که هر فیلسوف باید با دو چنگال غذا بخورد.

هر چنگال را فقط یک فیلسوف می‌تواند نگه دارد و همچنین تنها وقتی می‌تواند از چنگالی استفاده کند که در حال حاضر آن چنگال در اختیار فیلسوف دیگری قرار نداشته باشد. بعد از اینکه فیلسوف مدتی غذا خورد، باید چنگال‌ها را بر روی میز قرار دهد تا بقیه فیلسوف‌ها بتوانند از آن استفاده کنند. یک فیلسوف تنها می‌تواند چنگالهایی که در سمت چپ و سمت راستش قرار دارند را بردارد و تا وقتی که هر دو آنها را برنداشته، نمی‌تواند غذا خوردن را شروع کند. در این مسئله فرض می‌شود که بشقاب‌های ماکارونی هیچ‌گاه تمام نمی‌شوند و نامحدود هستند.

توجه داشته باشید که هر فیلسوف برای مدت زمان محدودی می‌تواند در حال خوردن باشد.

اگر فیلسوف‌ها به طور همزمان چنگال های سمت چپ یا راست خود را بردارند ، حالت بن‌بست یا Deadlock به وجود می‌آید که مشکل بسیاری از طراحان سیستم‌های همروند می باشد.

هدف مسأله این است تا استراتژی پیدا شود که :

- **حداقل اگر یک فیلسوف گرسنه شد بتواند غذا بخورد.**
- **میانگین غذاخوردن فیلسوف‌ها باهم برابر باشد.**

درشروع کار در نرم‌افزار نت لوگو هر فیلسوف به رنگ آبی است (در حال فکر کردن یا Thinking) . بعد از مدتی فیلسوف ممکن است گرسنه (hungry) شود پس در نتیجه رنگش به قرمز تغییر می کند. پس از اینکه فیلسوف گرسنه موفق شد چنگال سمت راست و چپ خود را تصاحب کند ، رنگش سبز می شود. پس :

رنگ فیلسوف در حال فکر کردن : آبی

رنگ فیلسوف گرسنه : قرمز

رنگ فیلسوف در حال خوردن : سبز

در صورتی که قابلیت همکاری یا *cooperation* فعال شود ، استراتژی به صورت زیر خواهد بود :

۱) اگر چنگال سمت چپ آزاد است ، آن را بردار.

۲) اگر چنگال سمت چپ در اختیار فیلسوف است ولی چنگال سمت راست آزاد نبود ، چنگال سمت چپ را آزاد کن

۳) اگر چنگال سمت راست آزاد است، آن را بردار

۴) اگر چنگال سمت راست در اختیار فیلسوف است ولی چنگال سمت چپ آزاد نبود ، چنگال سمت راست را آزاد کن

۵) اگر فیلسوف هم چنگال سمت چپ و هم چنگال سمت راست را دراختیار داشت می‌تواند شروع به خوردن نماید.

نحوه استفاده

- تنظیمات اولیه
  - *num-philosophers* : بیانگر تعداد فیلسوف‌های دور میز می باشد.

دکمه *Setup* برای تنظیم مسأله به حالت اولیه است. دکمه *go* نیز برای اجرای شبیه سازی و *go once* برای اجرای شبیه سازی به صورت مرحله به مرحله می باشد.

- **تنظیمات دیگر**
  - *hungry-chance* :بیانگر احتمال گرسنگی هر فیلسوف در هر مرحله می باشد.
  - *Full-chance* : بیانگر احتمال سیری هر فیلسوف در هر مرحله می باشد.
  - *Cooperation* : اگر غیرفعال باشد ، شبیه سازی به صورت ساده انجام خواهد شد در غیراین صورت از استراتژی همکاری که در بالا بیان شدِ، استفاده می شود.
- نمودار ها
  - *Spaghetti consumed* : میزان مصرف ماکارونی هر فیلسوف را نشان می دهد.
  - *Resource allocation* : وضعیت فیلسوف را نشان می دهد.

توضیحات مربوط به کدنویسی

Agents

جهان نت لوگو از یکسری عامل یا Agent تشکیل شده است که هرکدام موجوداتی هستند که می‌توانند طبق یکسری دستورالعمل عملیاتی را انجام دهند.هر عامل می بایست خودمختار – واکنش گرا – پیش فعال و اجتماعی باشد.

در نرم‌افزار نت لوگو چهارنوع عامل داریم :

- *turtles*
- *patches*
- *links*
- *observer*

به عامل هایی که قابلیت حرکت در جهان نت لوگو را دارند *turtles* گفته می شود. *World* یا جهان در نت‌لوگو دوبعدی است و شامل شبکه ای از تکه(*patches*)هاست.

هر *patch* یک یک عامل نامیده می‌شود که قابلیت حرکت ندارد و درواقع یک مربع از جهانی است که *turtels* می‌تواند در آن حرکت کنند.

.برای ارتباط بین دو *turtles* هم از عامل *link* کمک گرفته می‌شود

عامل *observer* یا ناظر موقعیت مکانی ندارد وآن را اینطور تصور کنید که *turtles* و *patches* را زیرنظر دارد. درواقع یکسری دستورات به سایر عامل ها می‌دهد.(همان کاربر)

وقتی جهانی در نت لوگو اجرا می‌شود هیچ *turtles* وجود ندارد وعامل *observer* است که می‌تواند *turtle* جدیدی بسازد.

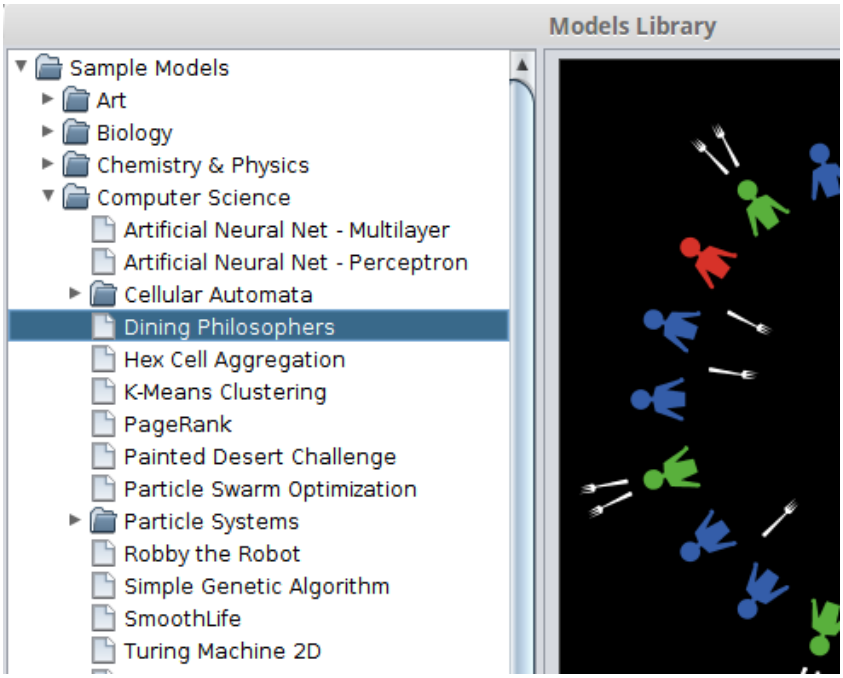
توجه داشته باشید که هر *patch* دارای مختصاتی است . مختصات اصلی (۰,۰) است و دیگر *patch* ها نسبت به این مختصات به صورت افقی و عمودی قرار می گیرند.

برای تنظیم یا برای فراخوانی مختصات یک *patch* از *pxcor* و *pycor* کمک گرفته می شود.

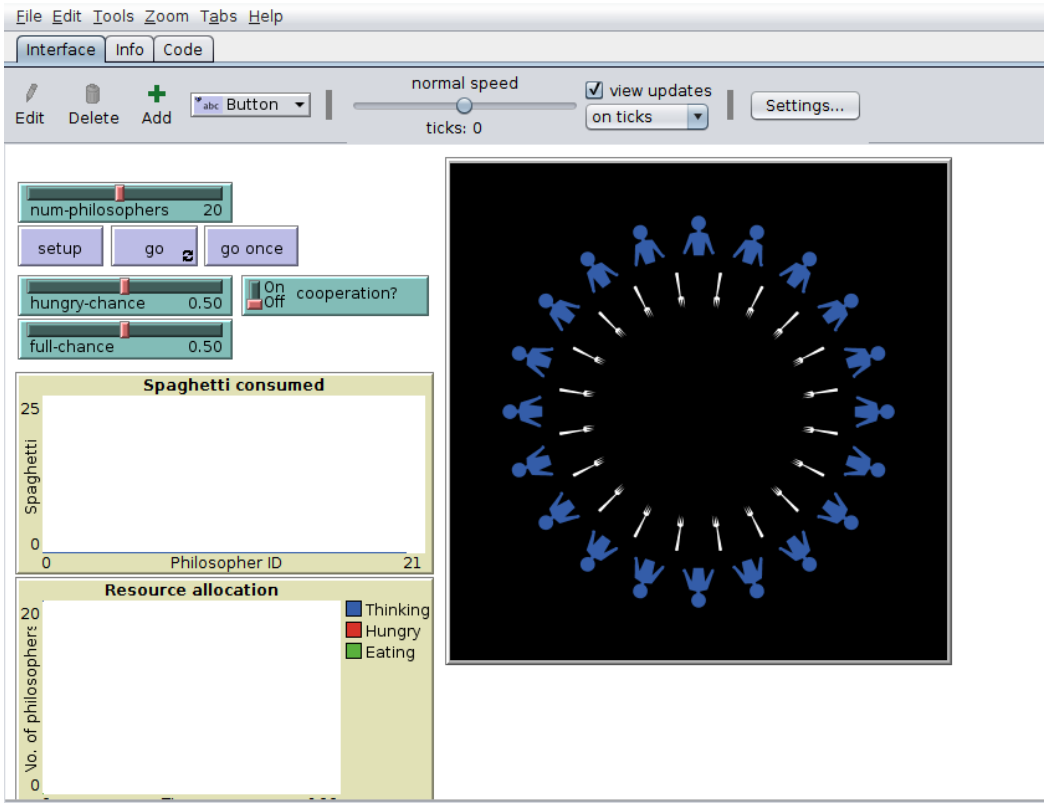
تعداد کل *patch*ها نیز با *min-pxcor* و *max-pxcor* و *min-pycor* و *max-pycor* مشخص می شود.

به این نکته نیز توجه داشته باشید که در هنگام اجرای یک جهان در نت لوگو min-pxcor و min-pycor برابر با ۱۶- و max-pxcor برابربا ۱۶ است.

از منوی *File→Models Library* را باز انتخاب کرده و سپس از میان مدل ها ، از شاخه *Computer Science* بر روی گزینه *Dining Philosophers* دوبار کلیک نمائید.



سپس در نرم‌افزار نت لوگو ،همانند تصویر زیر مدل مربوطه نمایش داده می شود.



حال از زبانه های موجود در زیر *MenuBar* تب *Code* را انتخاب کنید. می‌خواهیم کدهای این مدل را بررسی نمائیم.

Variables

هر عامل برای ذخیره مقادیری (مثل عدد یا نوشته) نیازمند متغیر یا متغیرهایی می باشد.

انواع متغیرها در نت لوگو عبارت است از :

۱. *Global variables* : همه عامل ها (*turtle* – *patch* و *link*) می‌توانند به مقدار این متغیر دسترسی داشته باشند. متغیر *Global* می بایست قبل از *Procedure* ها تعریف شود.

برای تعریف متغیر *Global* در نت لوگو از کلمه کلیدی *globals*[] استفاده می شود.

مثال : تعریف دو متغیر *global* به همراه توضیحات هر کدام.

```
globals [
   max_energy ;; maximum energy
   speed ;; traversed cells per simulation step
]
```

۲. **Agent variables** : هر عامل علاوه بر اینکه می‌تواند به متغیرهای عمومی یا *global* دسترسی داشته باشد نیز خودش دارای یکسری متغیرها می باشد.برای تعریف یک متغیر برای عامل های *turtle – patch – link* می‌توان به صورت زیر عمل کرد :

```
turtles-own [
   life
]
```

```
patches-own [
   grass_quantity
]
```

۳. **Local variables** :به متغیرهایی که داخل *procedure* ها معرفی می‌شوند گفته می شود.

همانطور که در مثال زیر مشاهده می‌شود *procedure* ای با نام *permuter* تعریف شده است که دارای دو آرگومان ورودی است.با کلمه کلیدی let یک متغیر محلی با نام tmp تعریف شده است و مقدار متغیر val داخل آن کپی می شود. در خط دوم مقدار متغیر *val2* در *val1* کپی می‌شود و در خط آخر نیز مقدار متغیر *tmp* در *val2* کپی می شود.

```
to permuter [val1 val2]
   let tmp val1
   set val1 val2
   set val2 tmp
end
```

breed

کلمه کلیدی *breed* برای تعریف یک نژاد در نت لوگو می باشد. در این مدل دو نژاد فیلسوف و چنگال تعریف شده است و هرکدام متغیرهای مربوط به خود را دارند.

بعد از کلمه کلیدی *breed* داخل براکت ، اولین مقدار به صورت جمع و دومین مقدار به صورت مفرد نوشته می شود.برای مثال *breed*[mice *mouse*]

```
breed [philosophers philosopher]
philosophers-own [
  state                 ;; my current state: "HUNGRY", "EATING", or "THINKING"
  left-fork right-fork  ;; the forks on my right and left
  total-eaten           ;; how much I've had to eat
]

breed [forks fork]
forks-own [
  home-xpos home-ypos home-heading     ;; where I belong when I'm on the table
  owner                                ;; the philosopher that currently owns me (if any)
  marked?                              ;; whether I'm currently marked
]
```

پس از تعریف نژاد یا breed برای هرکدام نیز متغیرهایی درنظر گرفته می‌شود که برای تعریف متغیر مربوط به هر عامل از دستور [... *var1*] *turtles-own* و برای هر نژاد از دستور [... *var1*] *breeds*-own استفاده می‌شود که به جای کلمه *breeds* نام نژاد موردنظر نوشته می شود.

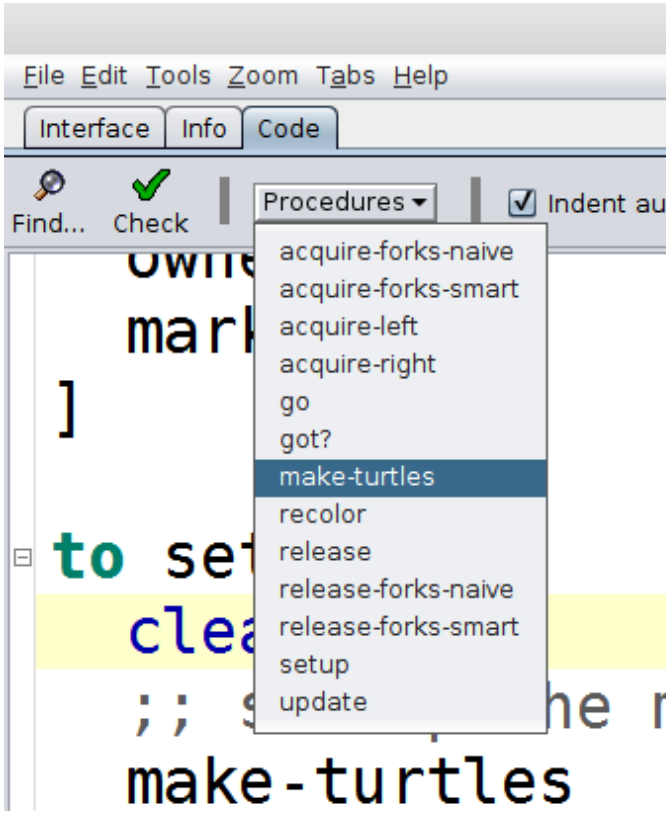
Setup Procedure

همانطور که می‌دانید تعریف Procedure در نت لوگو با to و end می باشد.

- ***clear-all*** : این دستور جهان نت لوگو را به حالت اولیه و طبق تنظیمات برمی گرداند. به جای نوشتن clear-all می‌توانید از مخفف شده آن یعنی ca نیز استفاده کنید.
- ***make-turtles*** و ***recolor*** هر دو *Procedure* می‌باشند که در ادامه توضیح داده می شوند.

```
to setup
  clear-all
  ;; set up the model
  make-turtles
  recolor
  reset-ticks
end
```

برای رفتن به دستورات هر Pr*ocedure* ؛ (که در اینجا می‌خواهیم به دستورات *make-turtles* دسترسی داشته باشیم)کافی است همانند تصویر زیر از بخش Procedures گزینه make-*turtles* را انتخاب نمائید.



Make-turtles Procedure

- ***set-default-shape*** : با استفاده از این دستور شکل مربوط به عامل و یا نژاد مورد نظر تعیین می شود. که در این مدل شکل مربوط به عامل فیلسوف شبیه آدمک و عامل چنگال دارای شکل چنگال می شود.
- ***create-ordered-philosophers*** و ***create-ordered-forks*** : با این دو دستور عامل فیلسوف و چنگال بـه تعداد خواسته شده ایجاد می شوند.

نحوه تعریف و استفاده از این دستور در نت لوگو به صورت create-ordered-turtles *number* می‌باشد که به جـای کلمـه *turtels* می‌تـوان نـژاد مربوطـه را نوشـت. همچـنین پس از دسـتور *create-ordered-turtles* ؛ *number* بیانگر تعداد عامل ها می باشد. *Num-philosophers* یک متغیر عمومی (*global*) می‌باشد و مقدار آن با توسط کاربر و با اسلایدر در زبانه *Interface* مشـخص می شـود.یکی از ویژگی‌هـای این دسـتور سـاخت عامل به صورت مرتب و به صورت دایره‌ای شکل از صفر تا ۳۶۰ می باشد. مخفف این دسـتور در نت لوگـو *cro* می‌باشد . برای مثال با این دستور *cro 5[fd 5]* در نت لوگو ۵ عامل به صورت مرتب ساخته می شود.

پس از مشخص شدن تعداد عامل در داخل براکت می‌توان دستوراتی را برای هرکدام مشـخص کـرد. همـانطور که در دستورات مربوط به عامل فیلسوف مشاهده می‌کنیـد پس از سـاخت آن ، انـدازه و وضـعیت مقـداردهی می‌شوند وهر عامل به میزان ۰.۳۵ پرش خواهند داشت.

```
to make-turtles
  set-default-shape philosophers "person torso"
  set-default-shape forks "fork"
  ;; create-ordered-<breed> equally spaces the headings of the turtles,
  ;; in who number order
  create-ordered-philosophers num-philosophers [
    set size 0.1
    jump 0.35
    set state "THINKING"
  ]
  create-ordered-forks num-philosophers [
    rt 180 / num-philosophers
    jump 0.25
    rt 180
    set size 0.1
    set marked? false
    set owner nobody
    ;; save my position and heading, so the philosophers can replace me later
    set home-xpos xcor
    set home-ypos ycor
    set home-heading heading
  ]
  ask philosophers [
    set left-fork fork (who + num-philosophers)
    ifelse who = 0
      [ set right-fork fork (2 * num-philosophers - 1) ]
      [ set right-fork fork (who + num-philosophers - 1) ]
  ]
  ask one-of forks [ set marked? true ]
end
```

- ***ask philosophers*** : با این دستور از فیلسوف‌ها می‌خواهیم تا عملیاتی را انجام دهند. شیوه تعریف این دستور به صورت ask *agentset* [*commands*] یا ask *agent* [*commands*] می‌باشد که می‌توان دستورات را برای

مجموعه‌ای از عامل ها یا نژاد ها و یا همه عامل ها درخواست کرد. برای مثال با دستور ask turtles [fd 1] از همه عامل ها می‌خواهیم تا یک واحد حرکت کنند ولی با دستور ask turtle 4 [rt 90] فقط عامل شماره ۴ به میزان ۹۰درجه چرخش خواهد داشت.

در این مدل می بایست متغیرهای *left-fork* و *right-fork* هر فیلسوف رامقداردهی شوندبرای مقدار دهی متغیر ها در نت لوگو از دستور *set* استفاده می شود.برای مثال *left-fork* مربوط به فیلسوف شماره ۲ برابر است با چنگال شماره ۵ که با دستور *set left-fork fork (who + num-philosophers)* به دست می آید.با استفاده *who* می‌توان id مربوط به هرعامل را پیدا کرد.

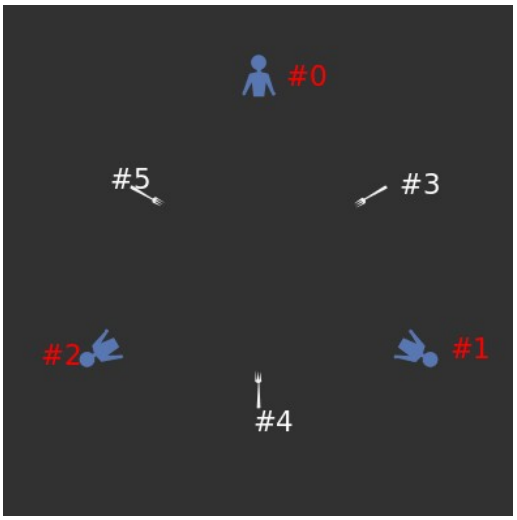
Left-fork :

#2 = set left-fork fork (who + num-philosophers) = set left-fork fork (2 + 3) = set left-fork fork(5)

Right-fork :

#2 = set right-fork fork (who + num-philosophers – 1) = set right-fork fork(2+3-1)=set right-fork fork(4)

*#0 = set right-fork fork (2 * num-philosophers – 1) = set right-fork fork(2*3-1) = set right-fork fork(5)*



- ***ask one-of forks*** : با این دستور از یکی از نژاد چنگال ها می‌خواهیم عملی را انجام دهد یا متغیری را مقداردهی کند.

recolor Procedure

در این متد از عامل ها خواسته می‌شود طبق شرایط مسأله رنگ خود را تغییر دهند. از عامل فیلسوف خواسته می‌شود اگر متغیر *state* آن برابر با *THINKING* بود رنگ خودش را آبی کند درغیراین صورت اگر درحال فکر کردن نبـود ولی گرسـنه (*HUNGRY*) بود رنگش قرمز شود درغیراین صورت رنگش سبز شود.

همچنین از عامل چنگال نیز درخواست شده است که اگر استراتژی *cooperation* فعال بود و همچنین متغیر *marked?* نیز دارای مقدار *true* بود ، رنگش ارغوانی و در غیراینصورت رنگش سفید شود.

```
to recolor
  ask philosophers [
    ;; look up the color in the colors list indexed by our current state
    ifelse state = "THINKING"
      [ set color blue ]
      [ ifelse state = "HUNGRY"
        [ set color red ]
        [ set color green ] ]
  ]
  ask forks [
    ;; we'll indicate marked forks only if cooperation is on.
    ifelse cooperation? and marked?
      [ set color magenta ]
      [ set color white ]
  ]
end
```

go Procedure

در این متد از یکی از فیلسوف‌ها خواسته شده است تا متد update را اجرا کند و پس از آن متد recolor فراخوانی می‌شود تا براساس شرایط مسأله رنگ عامل ها تغییر کند.

```
to go
  ask one-of philosophers [ update ]
  recolor
  tick
end
```

update Procedure

- ***random-float :*** در نت لوگو برای تولید یک عدد تصادفی از *random-float* استفاده می شود. برای مثال *random-float 3* در نت لوگو عددی تصادفی بزرگ‌تر از صفر و کوچکتر از عدد ۳ تولید می کند.

همانطور که در متد update مشاهده می‌شود اگر وضعیت عامل فیلسوف برابر با *THINKING* بود ، عددی تصادفی بین صفر تا یک تولید می‌شود و اگر آن عدد از متغیر عمومی *hungry-chance* کوچکتر بود ، متغیر *state* عامل برابر با گرسنگی (*HUNGRY*) می شود. اگر عامل گرسنه نبود و همچنان در حال فکر کردن بود با دستور *stop* خواسته می شود تا سایر شرایط بررسی نشود.

اگر فیلسوف در حال خوردن(*EATING*) بود به متغیر *total*-eaten یک واحد اضافه می شود و سپس عددی تصادفی بین صفر و یک تولید و با متغیر *full-chance* مقایسه می شود ؛ اگر مقدارش کوچکتر بود و نیز استراتژی cooperation نیز فعال بود ، استراتژی هوشمند(متد release-*forks*-smart) را انجام می دهد در غیراینصورت به صورت ساده متد -*release forks-naive* را انجام می‌دهدو نیز در آخر مقدار متغیر state برابر با *THINKING* می شود.

اگر فیلسوف گرسنه باشد ؛ بررسی می‌شود آیا استراتژی *cooperation* فعال است که در صورت فعال بودن متد -*acquire forks-smart* اجرا می‌شود و در غیراین صورت متد *acquire-forks-naive* اجرا خواهد شد. در پایان این وضعیت نیز از متد *got?* استفاده شده است که هریک از متدهای مذکر در ادامه توضیح داده شده است.

```
to update  ;; philosopher procedure
  if state = "THINKING" [
    if random-float 1.0 < hungry-chance [
      set state "HUNGRY"
    ]
    stop
  ]
  if state = "EATING" [
    ;; keep track of how much we're eating.
    set total-eaten (total-eaten + 1)
    if random-float 1.0 < full-chance [
      ;; put down forks
      ifelse cooperation?
        [ release-forks-smart ]
        [ release-forks-naive ]
      ;; continue thinking
      set state "THINKING"
    ]
    stop
  ]
  if state = "HUNGRY" [
    ; try to pick up the forks.
    ifelse cooperation?
      [ acquire-forks-smart ]
      [ acquire-forks-naive ]
    ; if we've got both forks, eat.
    if got? left-fork and got? right-fork
      [ set state "EATING" ]
    stop
  ]
end
```

release-forks-smart Procedure

در این متد بررسی می‌شود که آیا چنگال سمت چپ علامت دار است یا چنگال سمت راست؟!

اگر چنگال سمت چپ علامت دار بود (متغیر *marked*? آن برابـر بـا true بـود) مقـدار آن را برابـر بـا *false* می‌شـود و از چنگال سمت راست خواسته می‌شود تا متغیر *marked*? خود را برابر با true نماید.

اگر چنگال سمت راست علامت دار بود (متغیر marked? آن برابر بـا *true* بـود) مقـدار آن را برابـر بـا *false* می‌شـود و از چنگال سمت چپ خواسته می‌شود تا متغیر *marked*? خود را برابر با true نماید.

در پایان این متد نیز از متد *release* استفاده شده است.

```
to release-forks-smart  ;; philosopher procedure
  ;; check left fork
  ifelse [marked?] of left-fork [
    ask left-fork [ set marked? false ]
    ask right-fork [ set marked? true ]
  ]
  [
    ;; otherwise, check right fork.
    if [marked?] of right-fork [
      ask right-fork [ set marked? false ]
      ask left-fork [ set marked? true ]
    ]
  ]
  ;; release the forks.
  release left-fork
  release right-fork
end
```

release Procedure

این متد دارای یک آرگومان ورودی (چنگال) است. در این متد مالک آرگومان(چنگال) بررسی می‌شود در صورت داشتن مالک می بایست مالک آن چنگال به *nobody* تغییر پیدا کند.

```
to release [the-fork]  ;; philosopher procedure
  ask the-fork [
    if owner != nobody [
      set owner nobody
      setxy home-xpos home-ypos
      set heading home-heading
    ]
  ]
end
```

- ***setxy x y*** : برای تعیین مختصات x و y عامل از این دستور در نت لوگو استفاده می شود. برای مثال با دستور setxy 0 0 ، عامل به مکان صفر و صفر جا به جا می شود.

- ***set heading*** : برای چرخش عامل نیز از دستور heading استفاده می شود. برای مثال set heading 10+heading برابر با دستور rt 10 می باشد.

release-fork-naive Procedure

زمانی که استراتژی cooperation فعال نبود از این متد استفاده می شود.

```
to release-forks-naive  ;; philosopher procedure
  release left-fork
  release right-fork
end
```

acquire-forks-smart Procedure

این متد زمانی که استراتژی cooperation فعال باشد مورد استفاده قرار می گیرد. در این متد در ابتدای کار مالک چنگال سمت چپ بررسی می‌شود که آیا دارای صاحبی است یا خیر؟!

زمانی فیلسوف می‌تواند چنگال سمت چپ خود را بردارد که :

۱. چنگال سمت چپ علامت دار نباشد
۲. یا مالک چنگال سمت راست باشد.

زمانی فیلسوف می‌تواند چنگال سمت راست خود را بردارد که :

۱. چنگال سمت راست علامت دار نباشد.
۲. یا مالک چنگال سمت چپ باشد.

```
to acquire-forks-smart  ;; philosopher procedure
  if [owner] of left-fork = nobody [
    if ([not marked?] of left-fork) or got? right-fork
      [ acquire-left ]
  ]
  if [owner] of right-fork = nobody [
    if ([not marked?] of right-fork) or got? left-fork
      [ acquire-right ]
  ]
end
```

got? Procedure

در نت لوگو برای ساخت متدهایی که گزارش می‌دهند از to-report استفاده می شود. این متد برای بیان گزارش مالکیت آرگومان ورودی است که آرگومان آن چنگال می باشد.

```
to-report got? [the-fork]  ;; philosopher procedure
  report self = [owner] of the-fork
end
```

acquire-forks-naive Procedure

اگر استراتژی cooperation فعال نباشد ، فیلسوف برای برداشتن چنگال در هنگام گرسنگی از این متد استفاده می کند.به صورت خیلی ساده بررسی می‌کند اگر چنگال سمت چپ دارای مالک نبود آن را بردار و اگر چنگال سمت راست دارای مالک نبود نیز آن را بردار.

```
to acquire-forks-naive  ;; philosopher procedure
  if [owner] of left-fork = nobody
    [ acquire-left ]
  if [owner] of right-fork = nobody
    [ acquire-right ]
end
```

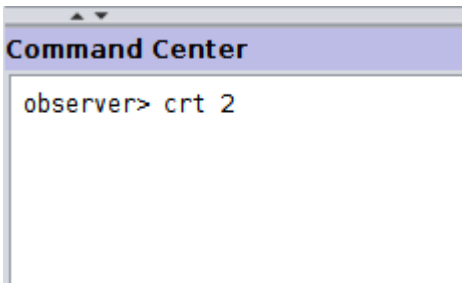
acquire-left Procedure – acquire-right Procedure

این متدها برای برداشتن چنگال سمت چپ و راست کاربرد دارند طوری که از هر کدام از چنگال ها خواسته می‌شودتا در تصاحب کدام فیلسوف قرار بگیرند. برای درک بهتر دستور move-to به مثال زیر توجه کنید :

```
to acquire-left  ;; philosopher procedure
  ask left-fork [
    set owner myself
    move-to owner
    set heading [heading] of owner
    rt 8  fd 0.1
  ]
end
```

مثال : می‌خواهیم دو عامل ایجاد کنیم و پس از حرکت هریک در جهان از یکی بخواهیم تا به مکان عامل دیگر حرکت کند.

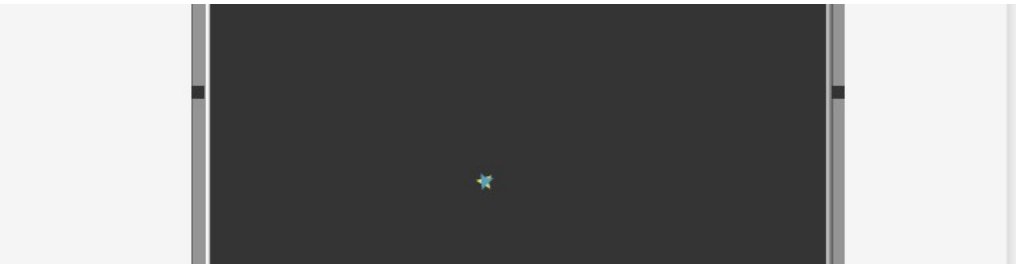
برای اجرای این مثال از بخش Command Center نرم‌افزار نت لوگو در پایین صفحه استفاده می‌کنیم . در حالت observer دستور crt 2 را وارد می‌کنیم که با این دستور دو عامل در جهان ایجاد می شوند.



حال از عامل ها می‌خواهیم تا هرکدام ۵ واحد حرکت کنند. که برای اینکار از دستور ask turtles [fd 5] استفاده می کنیم.

حال از عامل شماره صفر می‌خواهیم تا به مکان عامل شماره یک برود. برای اینکار از دستور move-to استفاده می کنیم.

ask turtle 0 [move-to turtle 1]



همانطور که مشاهده می‌شود عامل شماره صفر به مکان عامل شماره یک رفت.

Name : Mohammad Amin
Family :Abolhasannezhad
email : Info@itdn.ir
Mobile : 09154941383 - 09025610915