NetLogoでネットワーク科学

NetLogoでは，タートル同士をリンクでつないでネットワークをつくる事ができます.

これを利用して，ネットワーク上の進化ゲームのプログラムを作成し，実験してみます.

（なお，この資料はNetLogoでのプログラム作成（タートル・パッチの動かし方，インターフェイスの作り方など）について知識のある人を対象にしていま す.）

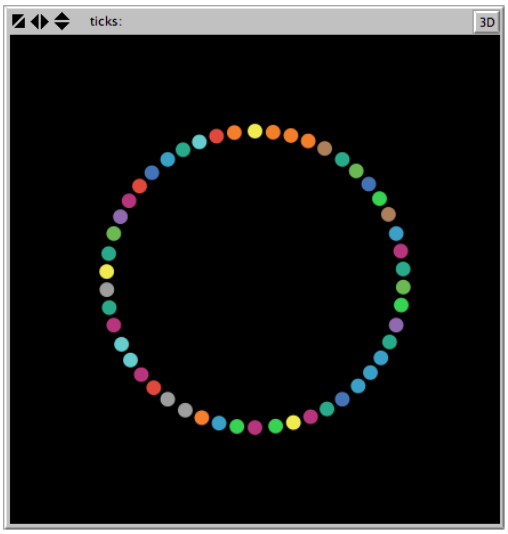
タートル同士をつなぐ

NetLogoにはリンクという特殊なエージェントがあり，それを使ってタートル同士を結合することで，ネットワークをつくることができます.

以下にいくつかの例を示します. ソースはそれぞれプロシージャを想定しているので，適宜実行するボタンなどを作成すること. また，変数名Nとしたスライダを 作成し，これを初期化時に生成するタートルの数とすること.

０）初期化してタートルをN体作成. 円周状に並べる.

```
to setup
   ca
   crt N
   ask turtles [
     set shape "circle"
   ]
   layout-circle sort turtles 10
end
```



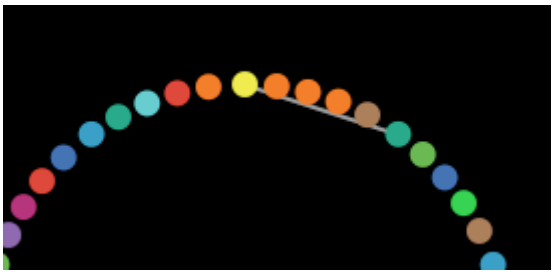
タートルには生成された順に通し番号が振られている. 上は番号順に並べるようにしてある.
番号nのタートルは(turtle n)で指定でき，タートルの持つ変数whoの中には自身の番号が入っている. （ask turtles [set label who]で番号を表示可能）

１）タートル0がタートル5につなぐ

```
to link-example1
    ask (turtle 0) [
        create-link-with turtle 5
    ]
end
```

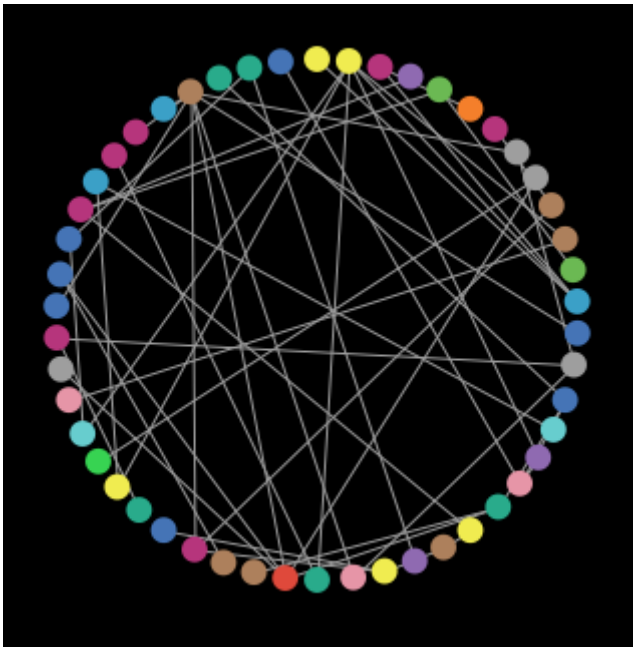
NetLogoではこのようにあるタートルに対して別のタートルにつなぐリンクを作らせるという書き方が基本です.

もし，リンクが細くて見にくい場合は適宜，"ask links [set thickness 0.2]"などとすると太くなります.

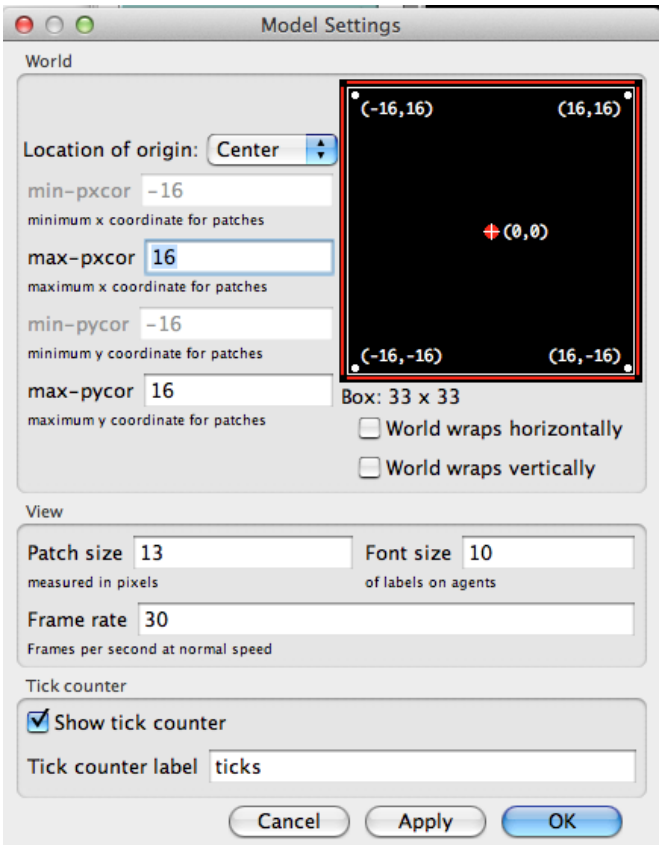


２）適当に一体選んで他の適当に選んだ未接続のタートルとつなぐ（平均次数２のランダムネットワーク）

```
to link-example2
    ask turtles [
        create-link-with one-of other turtles with [not link-neighbor? self]
    ]
end
```



このとき，右端と左端，上端と下端がつながってしまう場合は，フィールド画面を選択して右ボタンでメニューを出し，”World wraps horizontally"と"World wraps vertically"チェックボックスをオフにします.



3）バネモデルで整列

リンクをバネだと思ってビヨーンとやって整列

```
to link-layout-spring
    repeat 50 [ layout-spring turtles links 0.2 5 1 ]
end
```

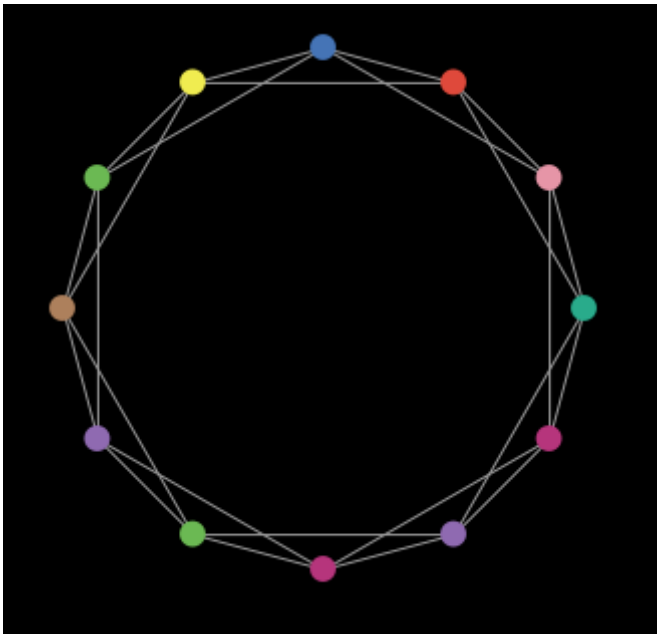
ネットワークの構造がより見やすくなります.

４）各タートルが両隣の二体と接続する（次数2のレギュラーネットワーク）

```
to link-example3
   ask turtles [
      create-link-with (turtle ((who + 1) mod N))
   ]
end
```

"a mod b"は，aをbで割ったあまり（C言語の%）を表します．（なぜこの処理が必要か考えてみましょう）

では，さらにその隣の2体と接続するには？（次数4のレギュラーネットワーク）



複雑ネットワーク

世の中には要素と要素がつながってネットワークをつくっている場合が多々あります．しかし，自然，社会，人工物など，この世界に存在するネットワークは，上 でつくったようなランダムに接続しているというより，ある種の特徴のあるネットワーク構造を持つ場合が少なからずあります.

そこで，そのような特徴を反映した２つの典型的なネットワークと，それを特徴付ける２つの指標について考えます.

スモールワールドネットワーク（βグラフ）

偶然であった人と共通の友人がいることがわかると，よく「世間は狭いですねぇ」と言うことがあります．このような人間関係には次の特徴があると言えます.

- 各人は近所同士など小さなグループの中で付き合いがちである

- しかし，ある人から遠いグループの別の人に至る友達の経路は意外に長くない

このような特徴を持つネットワークはスモールワールドネットワークと呼ばれ，人間の社会関係，ビジネスや組織の共同関係，Webページのつながり，遺伝子制 御の構造や細胞の代謝ネットワークなど，様々な場面に見られることが知られています.

スモールワールドネットワークの一つであるβグラフと呼ばれるもののうち，次数4のネットワーク（一つのノードから出ているリンク数が平均で4）は以下の手順で作 ることができます.

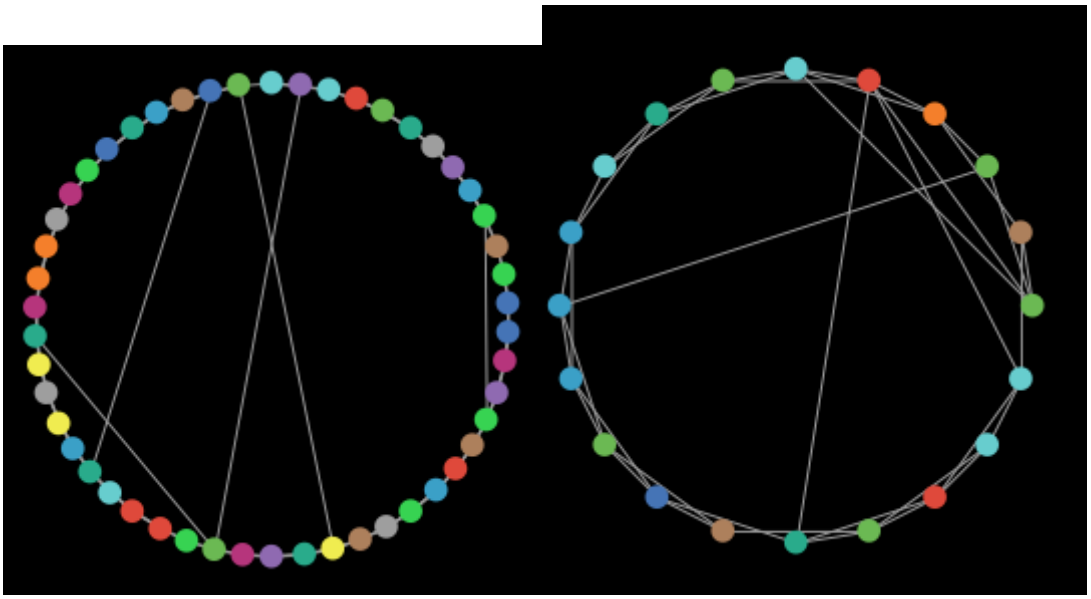
- 円周上にN個のノードが並んでいるとして，各ノードから右隣とその右隣（もしくは，左隣とその左隣）の二つのノードに向かって順にリンクでつないでい くとする.
- ただし，各リンクつくる際，ある確率PSで当初のノードにはつながず，ランダムに選んだ未接続のノードにつなぐとする.
- PSが小さいとき，ネットワークは上記のスモールワールド性を満たす.

練習）実際に次数4のスモールワールドネットワークを作るNetLogoのプログラムを自作してみましょう．ただし，今後のために，PSは0〜1の範囲で 0.005刻みで値が調整可能なスライダーの変数として作成しましょう.

ヒント：各ノードに対し，確率PSで処理Ａを実行し，確率1-PSで処理Ｂを実行するには，以下の様に書きます.

```
ask turtles [
    ifelse (random-float 1.0 < PS) [
        処理Ａ
    ][
        処理Ｂ
    ]
]
```

なお，円周上に配置されたノードの生成には従来のsetupプロシージャを使い，その後，ノード間に上記の手順でリンクを張るプロシージャをcreate-sw- linksとして定義することにします.



平均経路長とクラスタ係数

上記のように作っても，ネットワークにどんな特徴があるか構造を見ただけでははっきりしません．そこで，ネットワーク全体の持つ特徴を定量化する指標がいろ いろ提案されています．その代表例として次の二つを挙げます．

- 平均経路長：各ノードからそれ以外の各ノードに至る経路のリンク数の平均
- クラスタ係数：ネットワーク上の各ノードに接続された２つのノードの間にリンクが存在している割合（友達の友達が友達である割合）

実はこれを計算するのはそう簡単でないので，NetLogoのサンプルから拝借したコードを追加して，計算できるようにしましょう．

以下の変数の定義をソースの一番上に追加します．

```
globals [ clustering-coefficient average-path-length]
turtles-own [node-clustering-coefficient distance-from-other-turtles]
```

残りのプロシージャをソースのどこかに追加します．

```
to calc-clustering-coefficient
  ifelse all? turtles [count link-neighbors <= 1]
  [
    ;; it is undefined
    ;; what should this be?
    set clustering-coefficient 0
  ][
    let total 0
    ask turtles with [ count link-neighbors <= 1]
      [ set node-clustering-coefficient "undefined" ]
    ask turtles with [ count link-neighbors > 1][
      let hood link-neighbors
      set node-clustering-coefficient (2 * count links with [ in-neighborhood? hood ] /
                                       ((count hood) * (count hood - 1)) )
      ;; find the sum for the value at turtles
      set total total + node-clustering-coefficient
    ]
    ;; take the average
    set clustering-coefficient total / count turtles with [count link-neighbors > 1]
  ]
end

to-report in-neighborhood? [ hood ]
  report ( member? end1 hood and member? end2 hood )
end

to calc-average-path-length
  ;; reset the distance list
  ask turtles
  [
    set distance-from-other-turtles []
  ]
  let i 0
  let j 0
  let k 0
```

```
  let node1 one-of turtles
  let node2 one-of turtles
  let node-count count turtles
  ;; initialize the distance lists
  while [i < node-count]  [
    set j 0
    while [j < node-count]    [
      set node1 turtle i
      set node2 turtle j
      ;; zero from a node to itself
      ifelse i = j      [
        ask node1 [
          set distance-from-other-turtles lput 0 distance-from-other-turtles
        ]
      ][
        ;; 1 from a node to it's neighbor
        ifelse [ link-neighbor? node1 ] of node2        [
          ask node1 [
            set distance-from-other-turtles lput 1 distance-from-other-turtles
          ]
        ][
          ask node1 [
            set distance-from-other-turtles lput 99999 distance-from-other-turtles
          ]
        ]
      ]
      set j j + 1
    ]
    set i i + 1
  ]
  set i 0
  set j 0
  let dummy 0
  while [k < node-count]  [
    set i 0
    while [i < node-count]    [
      set j 0
      while [j < node-count]      [
        ;; alternate path length through kth node
        set dummy ( (item k [distance-from-other-turtles] of turtle i) +
                    (item j [distance-from-other-turtles] of turtle k))
        ;; is the alternate path shorter?
        if dummy < (item j [distance-from-other-turtles] of turtle i)        [
          ask turtle i [
            set distance-from-other-turtles replace-item j distance-from-other-turtles dummy
          ]
        ]
        set j j + 1
      ]
      set i i + 1
    ]
    set k k + 1
  ]

  let num-connected-pairs sum [length remove 99999 (remove 0 distance-from-other-turtles)] of
turtles
  ifelse ( num-connected-pairs != (count turtles * (count turtles - 1) ))
  [set average-path-length 0]
```

```
  [set average-path-length (sum [sum distance-from-other-turtles] of turtles) / (num-connected-pairs)]
end
```

calc-average-path-lengthを実行するとグローバル変数average-path-lengthに平均経路長が代入され，calc- clustering-coefficientを実行するとグローバル変数clustering-coefficientにクラスタ係数が代入されます．便利 ダネ！

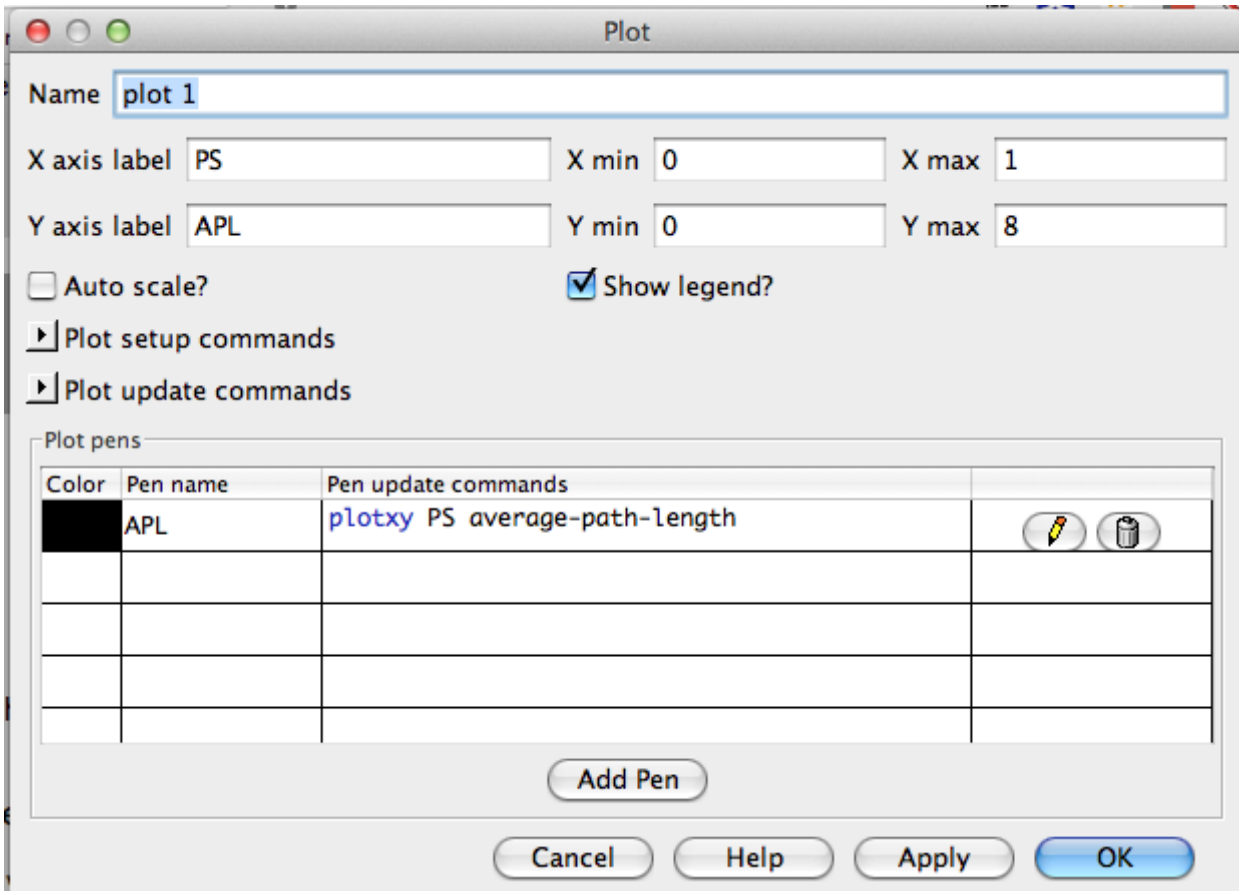
（注：上記のプロシージャはNetlogo Models LibraryのSmall Worldsモデルのソースコードを一部利用したものです．）

スモールワールドネットワークの特徴の分析

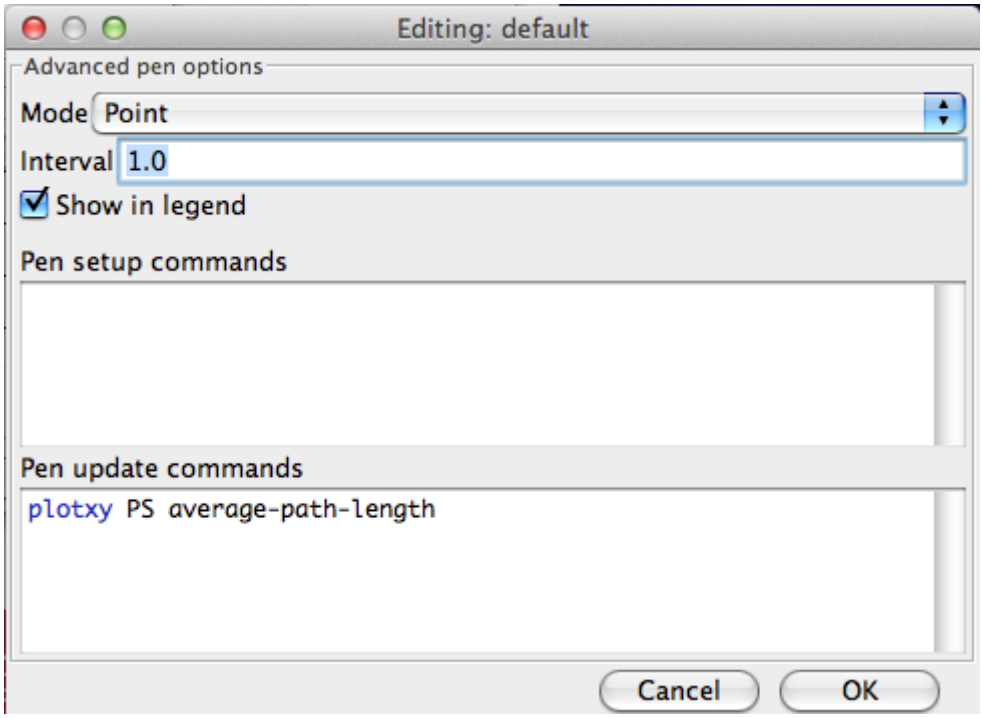
上のβグラフは，ランダムに接続する確率PSが0の場合レギュラーネットワークとなり，1の場合はランダムネットワークになります．PSの値と，出来上がったネッ トワークの持つ二つの指標との関係をグラフに表し，スモールワールドたる所以を探りたいと思います．

次のような拡張を行います．先ほどと同様の次数4のスモールワールドネットワークを考えます．PSを適当な値に設定してボタンを押すと，その設定でネットワークを 作り平均経路長が計算され，その結果が，横軸がPSの値，縦軸が平均経路長の値としたグラフにプロットされるようにします．

- インターフェイスウィンドウの左上のインターフェイスのリストから，”Plot”を選び，インターフェイス領域の空白のところでクリックして，グラフ を一つ作ります．
- グラフを右クリックしてメニューを出し，”Edit”を選び，以下のように入力します．



- さらに，上のウィンドウの鉛筆の部分をクリックして下のウィンドウを出し，以下のように設定します．



- 次のプロシージャを作り，実行するボタンも作ります（後の都合，setupは使わないことに注意）．

```
to create-calc-sw
  clear-turtles
  create-turtles N [
    set shape "circle"
  ]
  layout-circle sort turtles 10

  create-sw-links
  calc-average-path-length
  update-plots
end
```

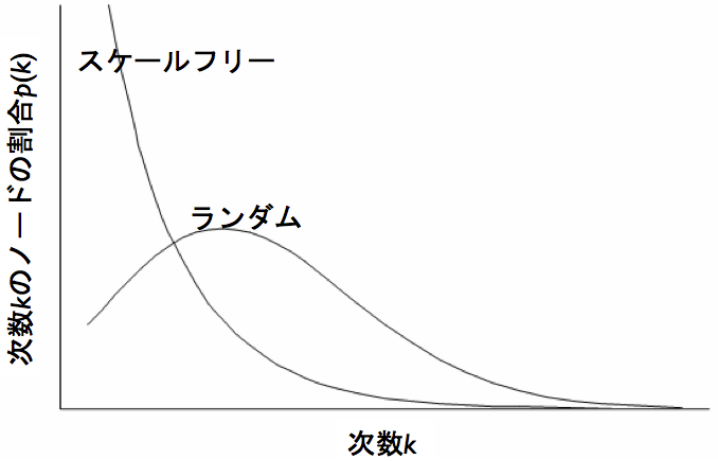
うまく設定できれば，PSの値をいろいろ変えながらcreate-calc-swを実行すると，そのときの平均経路長（APL）が次々とプロットされていき ます．

練習）クラスタ係数についても同時に計算し，その値を異なるグラフ，もしくは，同じグラフの上に異なるペンでプロットするようにグラフの設定を追加・変更し ましょう．同じグラフの場合はクラスタ係数を10倍するなど，値の大きさをそろえるとよいでしょう．

スケールフリーネットワーク

複雑ネットワークのもう一つの代表例が，スケールフリーネットワークです．各ノードの持つリンクの数をそのノードの”次数”と呼びます．スケールフリーと は，次数の大きいノードがごくわずかに存在する一方，次数が小さくなるに従いそのノード数が急激に増え，次数がわずかなノードが多数存在する性質のことをさし ます．横軸に次数，縦軸にその次数を持つノード数としたグ

ラフを描くと，値がべき乗則（∝1/次数^α）に従います．（ランダムネットワークの場合は二項分布 （ポアソン分布））



この次数分布のグラフはどこを切り取って拡大して見ても同じ形をしているので，スケールフリーと呼ばれています．このような特徴を持つネットワークも自然界や社会 においてよく見られることが知られています．

このようなネットワークを作る方法でよく知られたものに次のBA（ Barabási–Albert）モデルがあります．以下はその一例（平均次数が（ほぼ）４のスケールフリーネットワーク）です．

- 最初は3個のノードからなる全結合されたネットワークを考える．
- 既存の全ノードから，各ノードの次数に比例した確率で異なるノードを２つ選び，そのノードと新たに作成したノードを結合する．
- 上の手順を繰り返してノード数Nのネットワークを作る．このネットワークはスケールフリー性を持つ．

このネットワークを作成可能なように，以下のプロシージャを追加します．

select-turtle-paは，今あるすべてのノード（タートル）から，各ノードの次数に比例した確率でノードを一つ選んで返します．これを利用して 既存のノードに対してリンクを張ってスケールフリーネットワークを作成するプロシージャcreate-sfをつくります．

```
to-report select-node-pa
  let total random-float sum [count link-neighbors] of turtles
  let partner nobody
  ask turtles
  [
    let nc count link-neighbors
    if partner = nobody [
      ifelse nc > total
        [ set partner self ]
        [ set total total - nc ]
    ]
  ]
  report partner
end
```

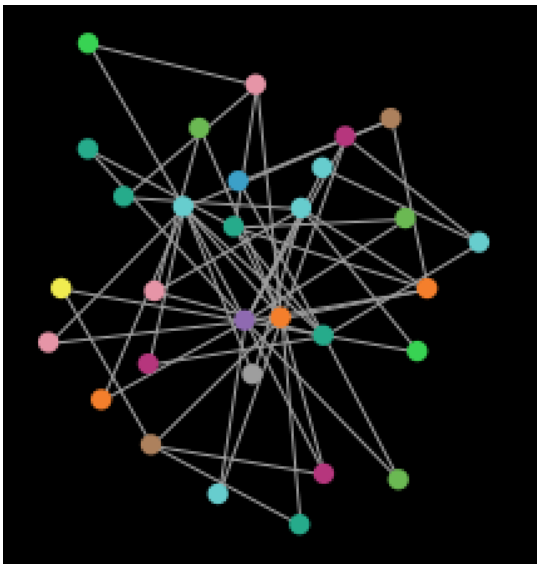
```
to create-sf
  ca
  crt 3 [set shape "circle"]
  ask turtle 0 [
    create-links-with other turtles
  ]
  repeat N - 3 [
      let partner select-node-pa
      let partner2 partner
      while [partner2 = partner][
        set partner2 select-node-pa
      ]
      create-turtles 1 [
        create-link-with partner
        create-link-with partner2
        set shape "circle"
      ]
  ]
    repeat 500 [ layout-spring turtles links 0.2 8 0.5 ]
    update-plots
end
```

現在のネットワークの次数分布（リンク数のヒストグラム）を表示するグラフを追加しましょう．ヒストグラムをプロットするには，ペンのコマンドを

histogram [count link-neighbors] of turtles

とし，ペンのモードを”Bar"にします.

練習）作成したネットワークがスケールフリー性を持つことを確認しましょう．また，これまで作成した他のネットワークの次数分布がどうなっているかも見てみ ましょう.



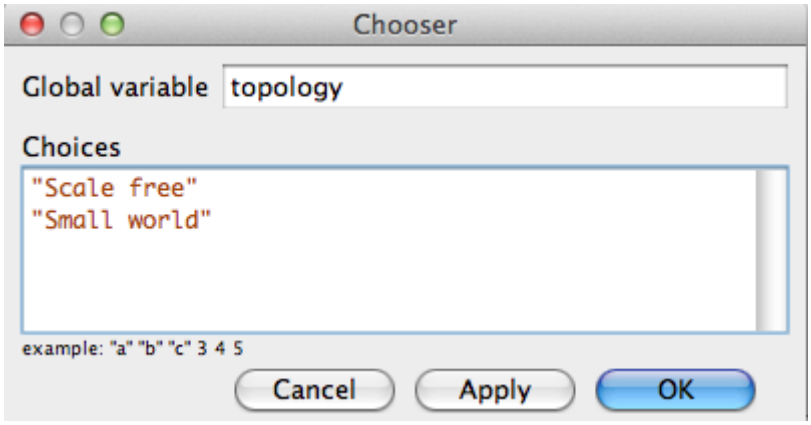
補足）現在のPSの値でスモールワールドネットワークをつくるcreate-swを以下の様に作成すると便利です.

```
to create-sw
    ca
    crt N [set shape "circle"]
```

```
    create-sw-links
    layout-circle sort turtles 10
    update-plots
end
```

また，いくつかの選択肢から一つ選ぶChooserというインターフェイスとif文を組み合わせると，選んだ項目に応じた処理ができます．

- インターフェイスのChooserを一つ作成する．このとき，以下のように設定したとする．



これは，グローバル変数としてtopologyを定義し，その値として，Scale freeかSmall worldの二つの値をインターフェイスのリストから選択できるようにすることを表す．

- たとえば，以下のようなプロシージャつくることで，topologyの値に応じて作成するネットワークを変更できる．

```
to setup_choose
  if (topology = "Scale free") [
    create-sf
  ]
  if (topology = "Small world") [
    create-sw
  ]
end
```

ネットワーク上のダイナミクス

こ れまでは，いくつかのネットワークとその特徴に注目してきましたが，ここからは，ネットワークの上で起きるいくつかの現象に注目します．

ネットワーク上の伝播

例えば，これまでに作成したネットワークが日常的な人付き合いの関係を表しているとします．つまり，つながっている同士は日常生活で出会う機会があるという ことです．

ここで，そろそろ寒くなってきたので，ある人が風邪をひいたり，インフルエンザにかかってしまったりしたとします．風邪ひきさんと出会うほど自分も風邪をひ きやすいとしたとき，人付き合いのネットワーク上で風邪はどのように伝播するでしょうか．

次のようなプロセスで表現してみましょう.

- 初期状態では，適当に選んだ一人が風邪をひいていて，他はすべて健康な人とする.
- 各ステップにおいて，各人は，ネットワーク上の近傍のランダムに選んだ一人と出会うとする．このとき，相手が風邪ひきだった場合，自分も風邪をひくと する．風邪は一度ひくとなおらないものとする.
- 上記のステップを繰り返す.

次の手順でプログラムを作ってみましょう.

- 各タートルは自分が風邪ひきかどうかを表す変数stateを持つ．0ならば風邪ひき，1ならば健康と見なす.
  turtles-own[...]の中に，stateを追加する.
  （フィールドのノードを右クリックしてインスペクタを開き，変数が追加されていることを確認してみましょう）
- 以下を実行する初期設定用のプロシージャsetup-propagate，画面更新用のプロシージャupdate-propagate，伝搬用のプロ シージャpropagateをソースに追加する.

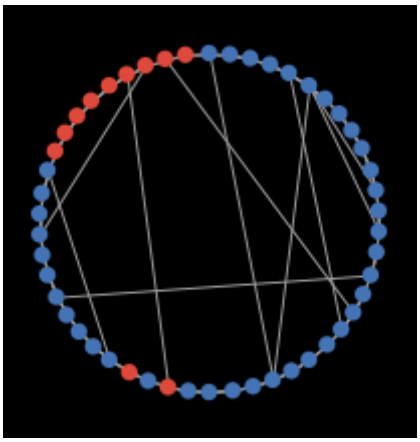
```
to setup-propagate
  setup_choose
  clear-all-plots
  ask turtles [
    set state 1
  ]
  ask one-of turtles [
    set state 0
  ]
  reset-ticks
  update-propagate
end

to propagate
  ask turtles [
    let partner one-of link-neighbors
    if (([state] of partner) = 0) [
      set state 0
    ]
  ]
  update-propagate
end

to update-propagate
  ask turtles [
    ifelse (state = 0) [
      set color red
    ][
    set color blue
    ]
  ]
  tick
end
```

setup-propagate，propagate実行用のボタンを作り，所期の動作になっているか確認しましょう.

さらに，次の手順で横軸をステップ，縦軸を風邪をひいた人の数を表すグラフを新たに追加しましょう.

- インターフェイスに新しいグラフを追加する.
- そのペンのコマンド設定を，"plot count turtles with [state = 0]"に設定する.

練習）スモールワールドネットワークにおいて，風邪がどのように伝播するかざっと見てみましょう.

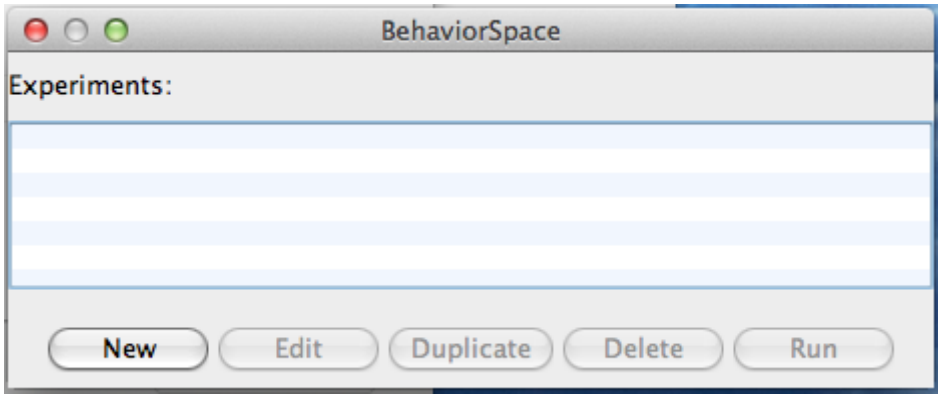


BehaviorSpaceでパラメータの影響を調べる

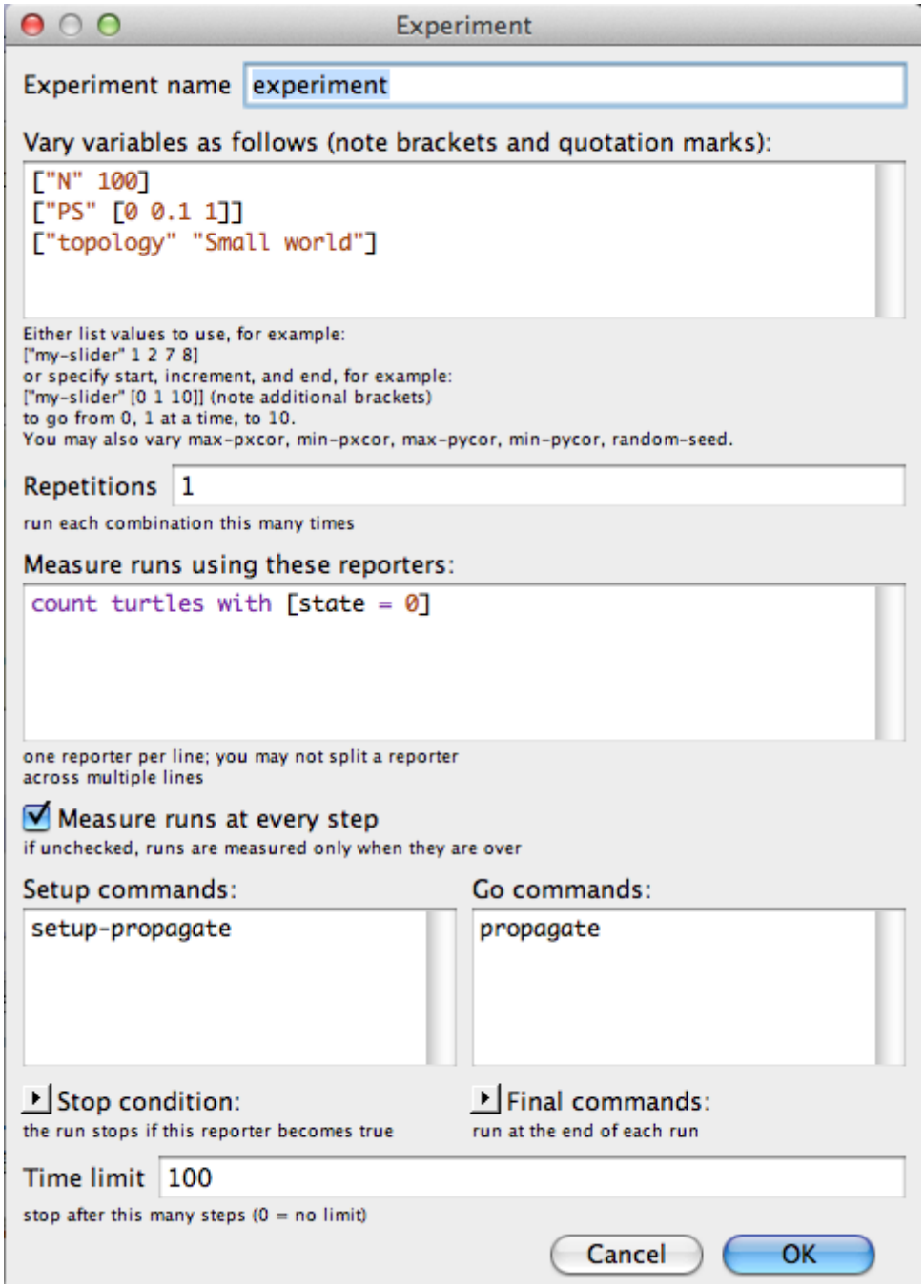
BehaviorSpaceという機能を使うと，パラメータ設定の違いが挙動に与える影響を比較するデータを簡単に作ることができます.

スモールワールドネットワークにおいて，PSの値を少しずつ変えた設定で実験したときに伝播の仕方がどう変わるかを示すデータを作ってみましょう.

- メニューから"Tools"→"BehaviorSpace"を選ぶ.
- 以下のウィンドウが出るので"New"を押して，実験設定を新しく登録する.



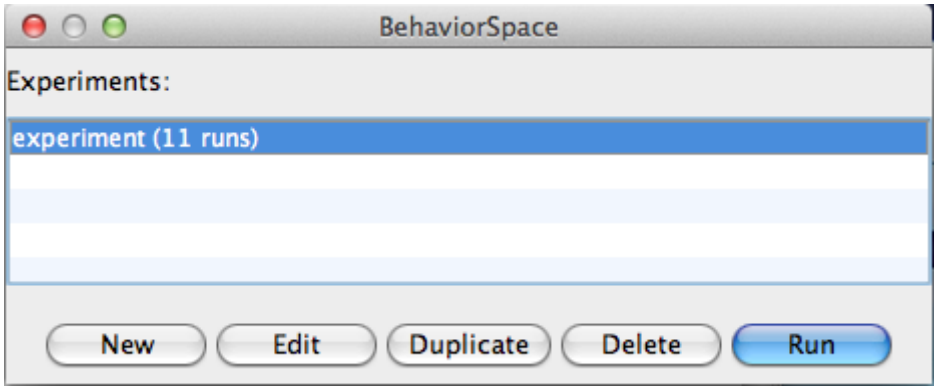
- さらに以下のウィンドウが出るので，この通りに設定する．



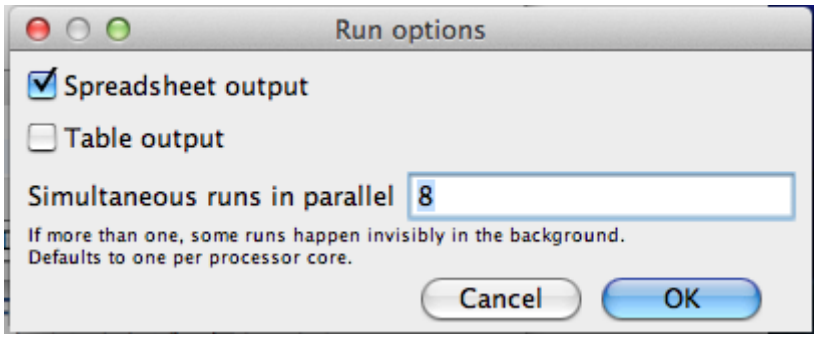
説明

- ["PS" [0 0.1 1]]："PSの値を0から1までの間で0.1刻みに増やした（11通りの）設定でそれぞれ実験する”ことを表す．["PS" 0 0.1 1]と，単に値を列挙すると，0, 0.1, 1の３つの設定でのみ実験を行うことに注意．
- Repetitions：各設定での繰り返し実験回数．
- Measure runs...：ここに，各ステップで出力する命令を一行ごとに書く．この場合，"count turtles with [state = 0]"なので，その状態でのsが0になっているタートルの数を出力する．
- Setup commands：各実験の初期設定のために実行される命令
- Go commands：各ステップで実行する命令
- Time limit：繰り返すステップ数
-

- 以上を設定し，以下の画面でexperimentを選択してRunを押す．



途中，実行の設定を聞かれるが，基本的にそのままOKで問題ない．



最後に出力ファイル名を指定する．デフォルトでソースのあるディレクトリに適当に名前を作って作成される．

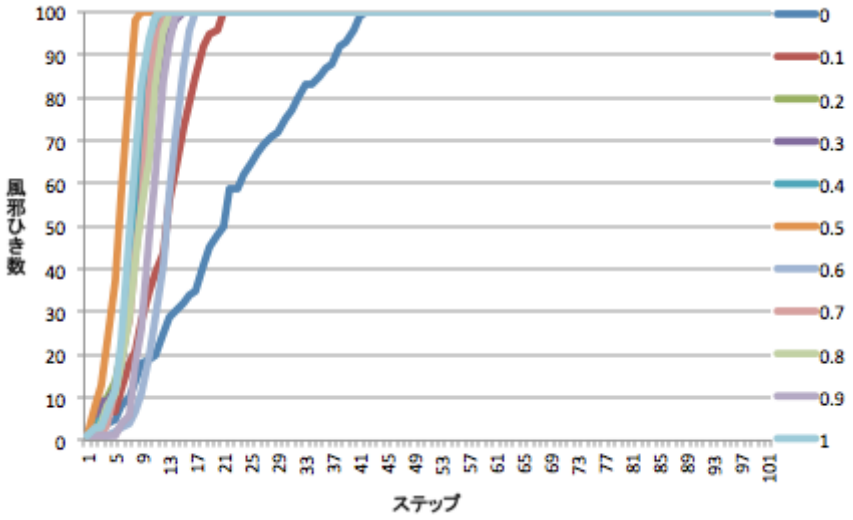
出力されるファイルの形式は，CSV（Comma separated vector）と呼ばれる，カンマ区切りのテキストファイル．エクセルなどの表計算ソフトで読み込むことができます．例えば以下のような感じ．

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	BehaviorSpace	results (NetLogo 5.0.4)											
2	network_rep1_sf_epi.nlogo												
3	experiment												
4	12/17/2013 16:54:37:887 +0900												
5	min-pxcor	max-pxcor	min-pycor	max-pycor									
6	-16	16	-16	16									
7	[run number]	1	2	3	4	5	6	7	8	9	10	11	
8	N	100	100	100	100	100	100	100	100	100	100	100	
9	PS	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	
10	topology	"Small world"	"Small world"	"Small world"	"Small world"	"Small world"	"Small world"	"Small world"	"Small world"	"Small world"	"Small world"	"Small world"	
11	[reporter]	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles with [state = 0]	
12	[final]	100	100	100	100	100	100	100	100	100	100	100	
13	[min]	1	1	1	1	1	1	1	1	1	1	1	
14	[max]	100	100	100	100	100	100	100	100	100	100	100	
15	[mean]	77.04950495	90.81188119	88.88118812	92.41584158	93.26732673	93.47524752	94.63366337	93.30693069	92.27722772	94.02970297	91.97029703	
16	[steps]	100	100	100	100	100	100	100	100	100	100	100	
17													
18	[all run data]	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles w	count turtles with [state = 0]	
19		1	1	1	1	1	1	1	1	1	1	1	
20		5	3	1	3	1	2	5	3	1	3	1	
21		5	6	4	5	1	5	11	6	1	6	1	
22		7	7	4	8	8	12	16	11	2	12	3	
23		9	14	7	12	16	23	30	14	8	25	6	
24		11	20	10	17	22	33	47	23	12	35	11	
25		15	24	13	26	35	42	66	40	23	53	19	
26		16	31	16	42	61	56	87	56	39	75	30	
27		17	42	19	59	81	79	96	81	59	91	51	
28		19	55	25	73	94	92	99	91	80	97	78	
29		22	67	35	91	100	97	100	98	94	99	94	
30		26	72	43	98	100	99	100	100	100	100	96	
31		28	77	56	99	100	100	100	100	100	100	98	
32		33	82	71	100	100	100	100	100	100	100	100	
33		35	87	84	100	100	100	100	100	100	100	100	
34		36	92	92	100	100	100	100	100	100	100	100	
35		37	95	96	100	100	100	100	100	100	100	100	
36		40	97	100	100	100	100	100	100	100	100	100	
37		44	100	100	100	100	100	100	100	100	100	100	
38		48	100	100	100	100	100	100	100	100	100	100	
39		48	100	100	100	100	100	100	100	100	100	100	
40		48	100	100	100	100	100	100	100	100	100	100	
41		51	100	100	100	100	100	100	100	100	100	100	
42		55	100	100	100	100	100	100	100	100	100	100	

説明

- 6行目まで：実験の設定などの基本情報
- 7行目：実験の通し番号（なので各列が各実験を表す）
- 8〜9行目：各実験で用いたグローバル変数の値
- 10行目：結果出力のコマンド
- 11行目：終了時の出力値
- 12行目：出力値の最小値
- 13行目：出力値の最大値
- 14行目：出力値の平均
- 15行目：実行したステップ数
- 16行目以降：各ステップでの出力値

このデータから，エクセルのグラフ化の機能を利用して，PSの影響に関するグラフを作ることができます.



ネットワーク上の進化

上の例は，”つながっていると広まる”という非常に単純なルールで動くネットワーク上のダイナミクスに関するものでしたが，感染症や噂など，意外と数多くの 場面に当てはまりそうです．

一方，もう一つ非常によくありそうなケースが，”良さそうなものをまねる（もしくは広がる）”というルールです．よく知った人が何かうまい方法を見つける と，それをまねしたくなるのは最もな状況の一つといえます．

こういった，”良さそうなものをまねる”という仕組みに基づいて生まれるダイナミクスを表現するのに，今回はゲーム理論の枠組みを活用します．

囚人のジレンマ

おそらくそこかしこで聞いたことがあると思いますが，囚人のジレンマゲームを利用します．これは，協力（Cooperate，C）もしくは裏切り （Defect，D）のいずれかの手を二人のプレイヤーが同時に出すゲームで，その結果，両者は以下の利得行列に示される利得を得ます．

↓P1 / P2 →	協力（C）	裏切り（D）
協力（C）	(R, R) = (4, 4)	(S, T) = (0, 5)
裏切り（D）	(T, S) = (5, 0)	(P, P) = (1, 1)

(x, y) = (P1の利得，P2の利得）

利得の値は，T, R, P, Sの組み合わせで定められ，T>R>P>Sを満たすとき，囚人のジレンマゲームと呼ばれます．このとき，以下の意味でジレンマが生じます．

- 相手が裏切ったと仮定しても，協力したと仮定しても，自分は裏切った場合の方が得（裏切りが支配戦略）
- なので，双方がこのように合理的に考えて行動すると，裏切り合いとなる（裏切り合いがナッシュ均衡）．
- しかし，互いに協力したほうが双方にとってメリットがある（裏切り合いがパレート最適でない）．

- なので，やっぱり協力すべきか，でも裏切られるかもしれない，うーんどっちかには決められないなぁ・・・というジレンマが生じる.

ネットワーク上のゲーム戦略の進化

では，このジレンマゲームの戦略，CかDを各人が持っていて，各ステップでネットワーク上の近傍の人とゲームを行い，その結果から，よりうまくやっている人の戦略 をまねるというルールで動くモデルを作ってみます.

ジレンマゲームでうまくいくかどうかは相手次第なので，相手を決めるネットワーク構造はその進化になんらかの影響があることが推測されます.

次の手順で作成します.

- 上で利用した各ノードが持つ変数stateを，今度はジレンマゲームの戦略を表す変数として利用する．state=0なら裏切り，state=1なら 協力と見なすことにする.
- 各ノードがゲームをした結果得られる得点を順次足していく変数scoreを定義する.
  （turtles-own[...]の中にscoreを追加）
- 次のステップでの戦略を保持する変数state_nextを各ノードが持つように定義する.
- R, T, P, Sをそれぞれスライダーの変数として定義し，その定義域を0から5まで，0.01刻みで設定する.
- 次のことを行うプロシージャgameを追加する．各ノードについて，その近傍のノードとゲームをしたときの合計の得点を計算し，scoreに保存す る.

```
to game
  ask turtles [
    set score 0
  ]
  ask turtles [
    let player self
    ask link-neighbors [
      if (([state] of player = 0) and (state = 0)) [
        set score (score + P)
      ]
      if (([state] of player = 0) and (state = 1)) [
        set score (score + S)
      ]
      if (([state] of player = 1) and (state = 0)) [
        set score (score + T)
      ]
      if (([state] of player = 1) and (state = 1)) [
        set score (score + R)
      ]
    ]
  ]
end
```

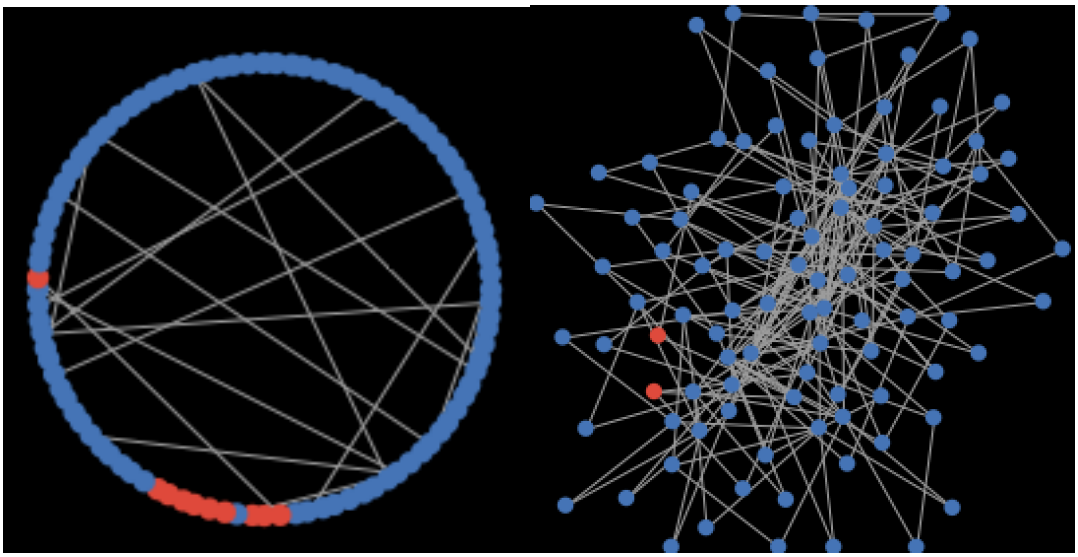
- 次のことを行うプロシージャevolveを追加する．各タートルが上記の通りにゲームを行い平均得点を計算したあと，自身のscoreと近傍のものを 比べ、近傍のものが自身のものより大

きい場合には、そのうち最大のscoreを持つタートル（複数いればランダムに選んだ）の戦略を次のステップでの自分 の戦略とする.

```
to evolve
  game
  ask turtles [
    set state_next state
  ]
  ask turtles [
    set state_next state
    if((max [score] of link-neighbors) > score) [
      set state_next ([state] of (max-one-of link-neighbors [score]))
    ]
  ]
  ask turtles [
    set state state_next
  ]
  update-propagate
end
```

- 適宜上記のプロシージャを組み合わせてプログラムが動くようにする.

setup-propergateでモデルを初期化し，evolveボタンを押すたびに戦略の分布が変 化していくことを確認しましょう.



N=100の場合