INTRODUCTION TO NetLogo

Rene Doursat
Department of Computer Science & Engineering
University of Nevada, Reno

翻訳&改訂　倉橋節也

Introduction to NetLogo

- NetLogoとは何か
  - 複雑システムのモデリング
  - 歴史
  - NetLogoの世界
- グラフィカルインターフェース
- プログラミングコンセプト
- チュートリアル

NetLogoとは何か

- 自然と社会現象をシミュレーションするプログラミング環境
  - 進化する複雑システムをモデリングできる
  - 同時実行する数100～数1000のエージェント
  - 個人のミクロレベル行動と、創発するマクロレベルパターン間の関連を探索する

NetLogoとは何か

- 扱いやすいアプリケーション開発環境
  - シミュレーションを開始実行
  - カスタムモデルの生成：自己組織化システムの仮説検証
  - モデルライブラリ：自然・社会科学での膨大なコレクション
  - シンプルなスクリプト言語
  - ユーザフレンドリーなグラフィカルインターフェース

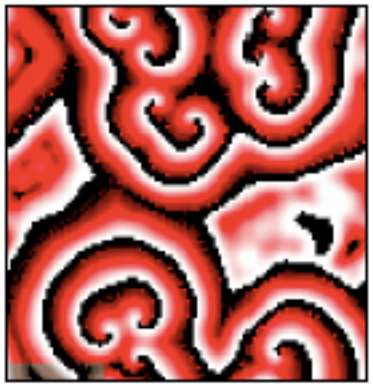
簡単な歴史

- Logo(1967)
  - LISP系の教育向け言語
- StarLogo(1991)
  - エージェントベースシミュレーション言語
- NetLogo(1999)
  - 連続タートル、座標、クロスプラットフォーム、ネットワーク

NetLogoの世界

- 3種類のエージェントによる2D/3D世界
  - patch：背景を作る
  - turtle：patchの上を移動する
  - observer：世界での全ての実行を観察する

  - Patchの例

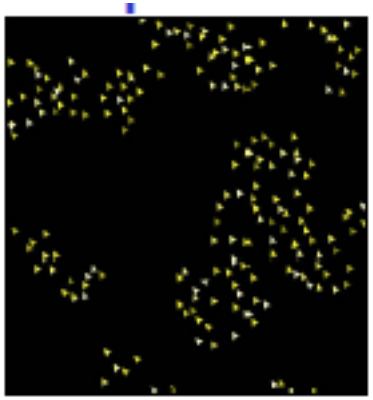


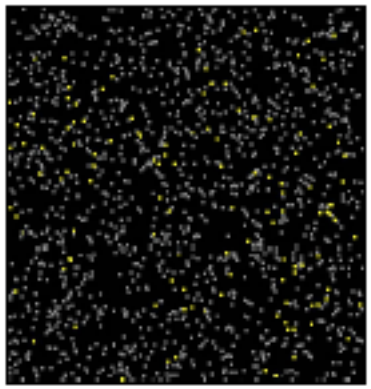
B-Z reaction

Fur

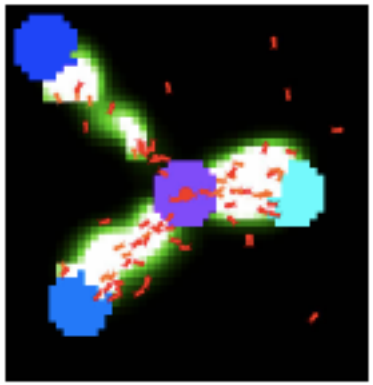
NetLogoの世界

- Turtleの例


Flocking


Fireflies

- Patch & turtleの例


Ants

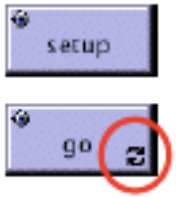

Termites

Introduction to NetLogo

- NetLogoとは何か
- グラフィカルインターフェース
  - コントロール
  - 設定
  - ビュー
- プログラミングコンセプト
- チュートリアル

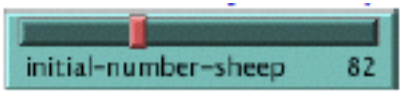
グラフィカルインターフェース

- コントロール（青） 実行フローを制御
  - ボタン、コマンドセンター
- ボタン（初期化、開始、停止、ステップ実行）
  - once ボタンは一度だけ実行
  - foreverボタンは繰り返し実行

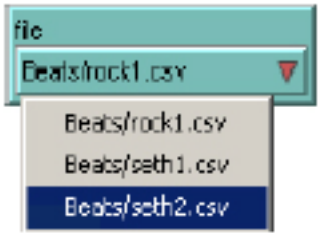


- コマンドセンター
  - ask observer/patches/turtles
  - 実行中に特定の命令をだせる

グラフィカルインターフェース

- 設定（緑） パラメータを設定・変更
  - スライダー、スイッチ、選択
- スライダー
  - 

initial-number-sheep = 82
- スイッチ
  - 

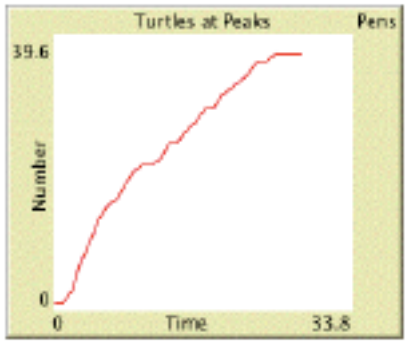
show-energy? = false
- 選択
  - 

file = “Beats¥seth2.csv”

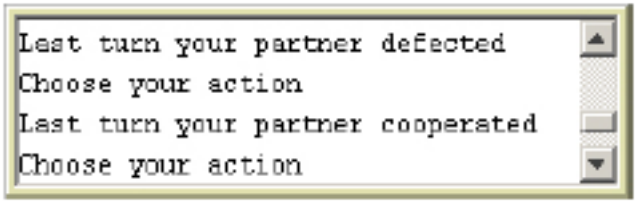
グラフィカルインターフェース

- ビュー（ベージュ） 情報の表示
  - モニター、プロット、テキスト、グラフィックス窓
- モニター　変数の現在値を表示

- プロット　変数の時系列を表示



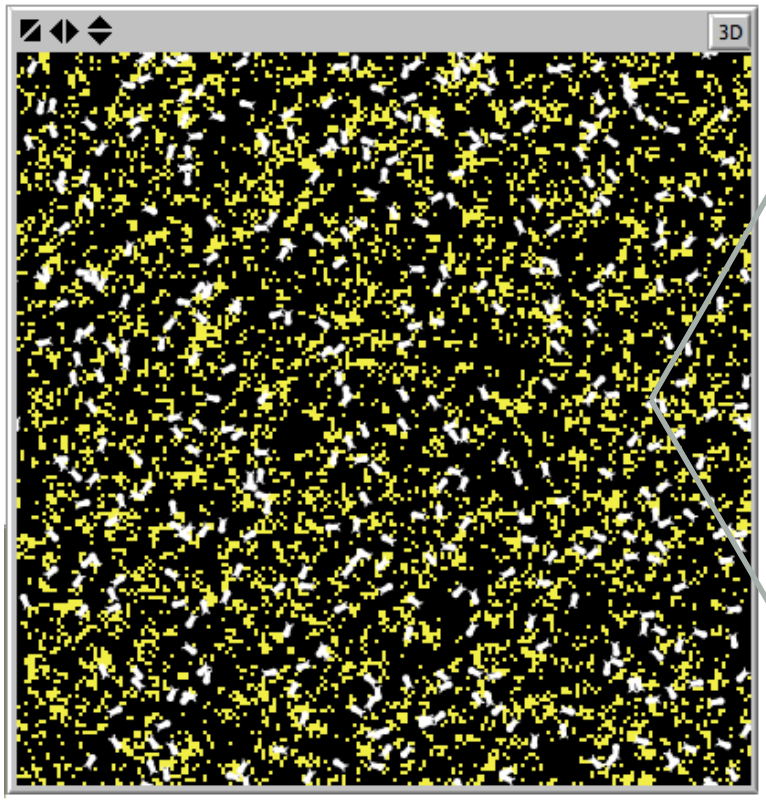
テキスト出力エリア



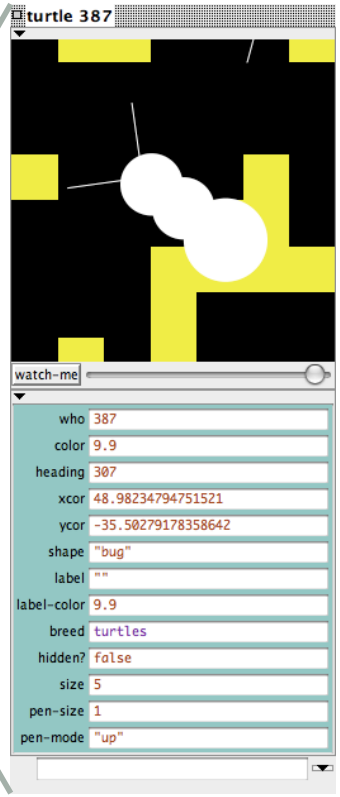
グラフィカルインターフェース

- グラフィカルウィンドウ　主たるビュー

幅と高さの変更



右クリックして
タートル／パッチの検査



Introduction to NetLogo

- NetLogoとは何か
- グラフィカルインターフェース
- プログラミングコンセプト
  - agents
  - procedures
  - variables
  - ask
  - agentsets
  - breeds
  - syncronization
- チュートリアル

プログラミング agents

- agents 自身の活動を同期実行
  - patches
    - 移動しない
    - 整数の座標を持つ(pxcor, pycor)
  - turtles
    - パッチの上を移動する
    - 小数点の座標(xcor,ycor)と方向(heading)を持つ
  - observer
    - 新しいタートルを作れる
    - 全てのagentと変数にアクセスし読み書きできる

プログラミング procedures(手続き)

- command
  - void functionの例

```
to setup
  clear-all
  create-turtles 5
end
```

  - 2つの引数の例

```
to go
  draw-polygon 5 7
end

to draw-polygon [num-sides p-size]
  pen-down
  repeat num-sides
    [ forward p-size
      right (360 / num-sides) ]
end
```

プログラミング procedures(手続き)

- reporter
  - 結果の値をレポート(リターン型関数)
  - 1引数の例

```
show absolute-value 2
show absolute-value -3

to-report absolute-value [number]
  ifelse number >= 0
    [report number]
    [report 0 - number]
end
```

プログラミング variables(変数)

- 変数(variables) 値を格納する場所
  - グローバル変数
    - ただひとつの値
    - すべてのエージェントからアクセスできる
  - タートル、パッチ変数
    - 各タートル/パッチが値を持つ
  - ローカル変数
    - プロシージャ内でのみアクセスできる
    - スコープ：最近傍の[ ]内、あるいはプロシージャ内
  - 組み込み変数 build-in variables
    - turtle変数：color, xcor, ycor, heading, etc.
    - patch変数：pcolor, pxcor, pycor, etc

プログラミング variables(変数)

- 変数定義
  - グローバル変数

```
globals [clock]
```

  - タートル/パッチ変数

```
turtles-own [energy speed]
patches-own [friction]
```

  - ローカル変数

```
to swap-colors [turtle1 turtle2]
  let temp [color] of turtle1
  let temp2 [color] of turtle2
  ask turtle1 [set color temp2]
  ask turtle2 [set color temp]
end
```

プログラミング variables(変数)

- 変数の設定
  - 全タートルに色を設定

```
ask turtles [set color red]
```

  - 全パッチに色を設定

```
ask patches [set pcolor red]
```

  - タートルの下のパッチに色を設定

```
ask turtles [set pcolor red]
```

  - 一つのタートルに色を設定

```
ask turtle 5 [set color green]
```

  - 一つのパッチに色を設定

```
ask patch 2 3 [set pcolor green]
```

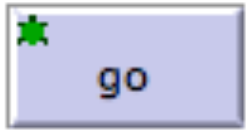
プログラミング ask(要求)

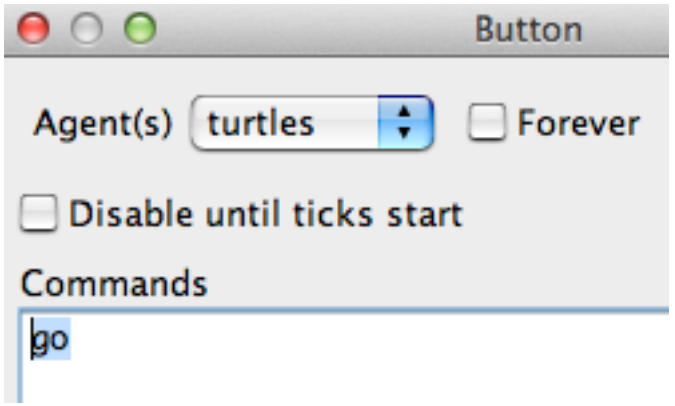
- ask コマンドの実行先の特定

```
ask turtles [fd 50]
ask turtle 5 [set color green]
ask patches [set pcolor red]
ask patch 2 3 [set pcolor green]
```

- ボタンでagentタイプを指定した場合はask不要

```
to go
  set color red
end
```





- オブザーバコードにはaskは付けない

```
clear-all
create-turtles 5
```

プログラミング agentsets(agent集合)

- agentset　agentの部分集合
  - すべての赤いタートル
    ```
    turtles with [color = red]
    ```
  - 現在の呼び出し者のパッチ上の赤いタートル
    ```
    turtles-here with [color = red]
    ```
  - 画面の右側にいるパッチ
    ```
    patches with [pxcor > 0]
    ```
  - 呼び出し者から3パッチ以内のタートル
    ```
    turtles in-radius 3
    ```
  - 呼び出し者の東西南北の4パッチ
    ```
    patches at-points [[1 0] [0 1] [-1 0] [0 -1]]
    neighbors4
    ```

プログラミング agentsets(agent集合)

- agentsetの使い方
  - コマンド実行

```
ask turtles with [color = red] [set color blue]
```

  - 存在するかどうかのチェック

```
show any? turtles-here with [color = red]
```

  - agent数のカウント

```
show count patches with [pxcor > 0]
```

  - 例:最富豪の排除

```
ask max-one-of turtles [sum assets] [die]
```

プログラミング breeds(種類)

- breed タートルの種類
  - 定義

```
breed [wolves wolf]
breed [sheep a-sheep]
```

  - 生成

```
create-wolves 10
```

  - そのbreedだけが持つタートル変数

```
wolves-own [age]
```

  - breedはタートル変数の一つ、breedを変更することができる

```
ask turtle 5 [if breed = wolves [set breed sheep]]
```

プログラミング syncronization(同期)

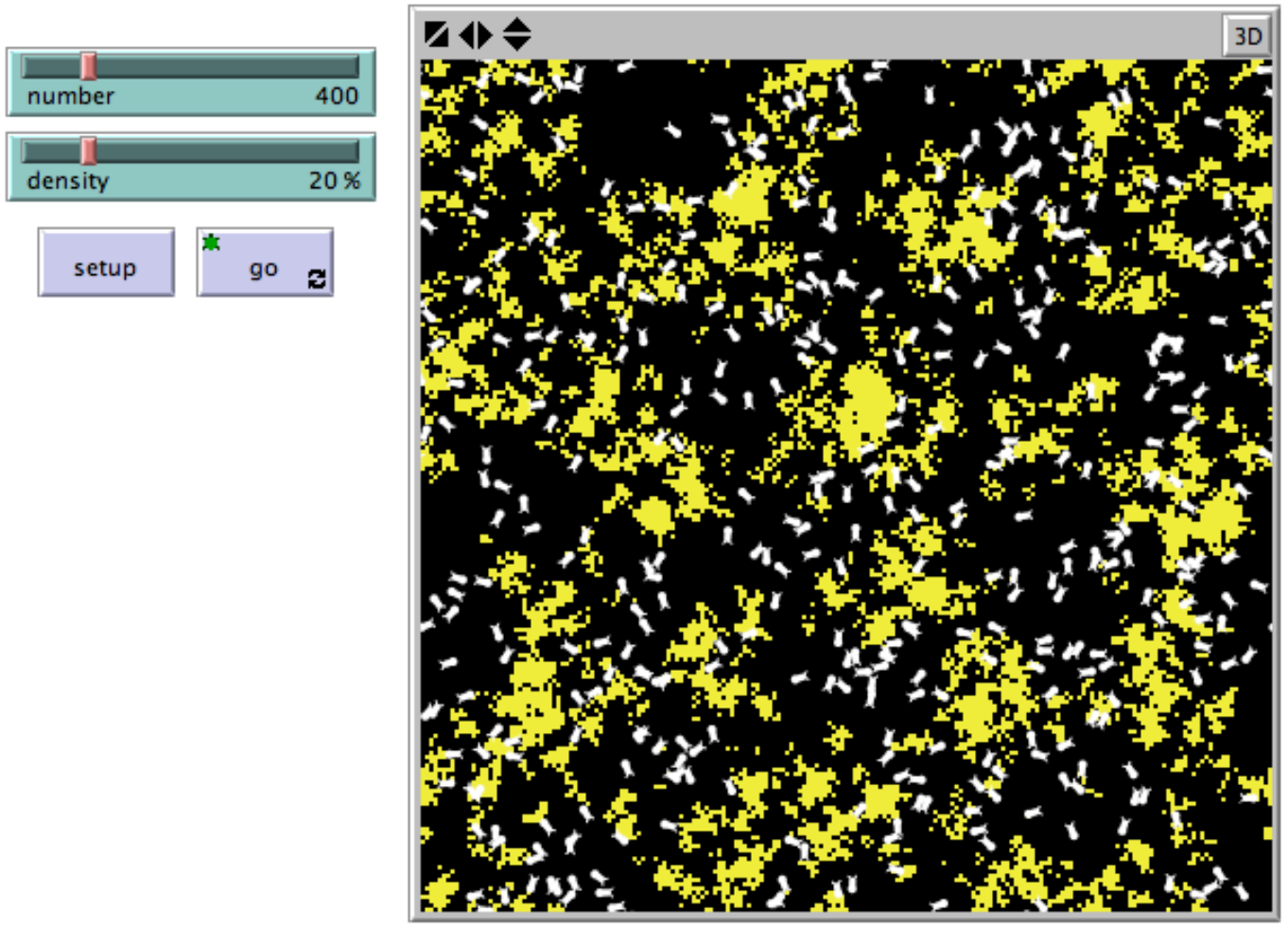
- 各agentは独立したスレッドで非同期実行

```
ask turtles [ fd random 10
              do-calculation ... ]
```

- 各ブロックの終了に合わせて同期実行

```
ask turtles [ fd random 10 ]
ask turtles [ do-calculation ] ...
```

Termits（シロアリ）のモデル



Termits : setup

```
to setup
  clear-all
  set-default-shape turtles "bug"
  ;; randomly distribute wood chips
  ask patches
  [ if random-float 100 < density
    [ set pcolor yellow ] ]
  ;; randomly distribute termites
  create-turtles number [
    set color white
    setxy random-xcor random-ycor
    set size 5  ;; easier to see
  ]
end
```

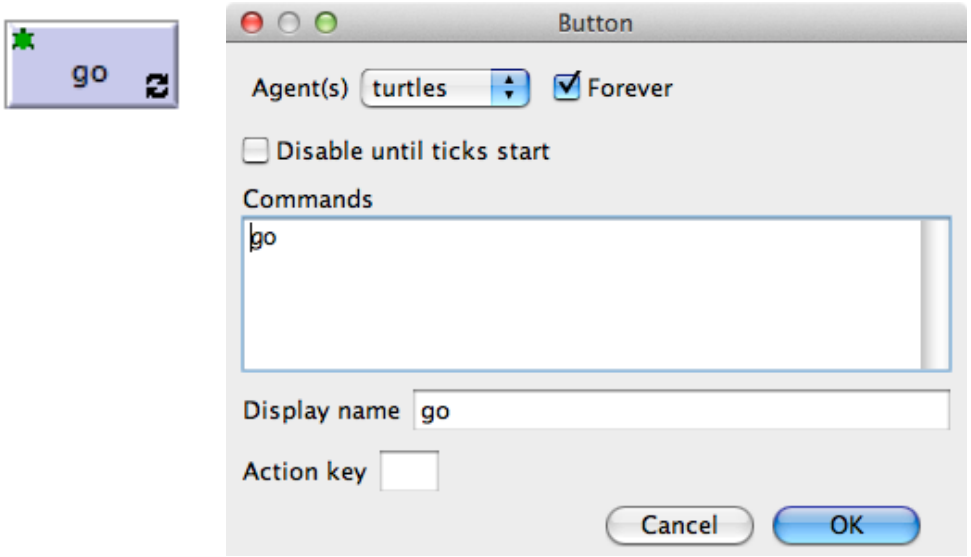
random-floatは実数値乱数を生成
densityはスライダーで設定

タートル数(number)はスライダーで設定

Termits : 実行　go

```
to go  ;; turtle procedure
  search-for-chip
  find-new-pile
  put-down-chip
end
```

goボタンで、3つのプロシージャを繰り返し実行
- 木片を探す
- 木杭を探す
- 木片を降ろす



Termits : 木片を探す　find-new-chip

```
to search-for-chip  ;; turtle procedure -- "picks up chip" by turning orange
  ifelse pcolor = yellow
  [ set pcolor black
    set color orange
    fd 20 ]
  [ wiggle
    search-for-chip ]
end
```

パッチ色が黄なら（木片を拾う）
　パッチ色を黒、タートル色をオレンジに変更し、
　20進む
パッチ色が黄でなければ、
　小刻に移動(wiggle)し、木片を探す(search-for-chip)

※search-for-chipが再帰的になっていることに注意
　黄だったら（拾って）黒くして20進んで次のプロシージャへ

Termits：木杭を探す　find-new-pile

```
to find-new-pile  ;; turtle procedure -- look for yellow patches
  if pcolor != yellow
  [ wiggle
    find-new-pile ]
end
```

パッチ色が黄でなければ、
　小刻に移動(wiggle)し、木杭を探す(find-new-pile)
　黄色（木杭）だったら次へ

※find-new-pileが再帰的になっていることに注意
　黄だったら次のプロシージャへ

Termits : 木片を降ろす　find-new-pile

```
to put-down-chip  ;; turtle procedure -- finds empty spot & drops chip
  ifelse pcolor = black
  [ set pcolor yellow
    set color white
    get-away ]
  [ rt random 360
    fd 1
    put-down-chip ]
end
```

パッチ色が黒なら、
　パッチ色を黄、タートル色を白に変更し、
　離れる(get-away)
パッチ色が黒でなければ（先にこちらを実行）、
　方向をランダムに0～360度回転し、
　1進んで
　木片を降ろす (put-down-chip)

※put-down-chipが再帰的になっていることに注意

Termits : 離れる　get-away

```
to get-away  ;; turtle procedure -- escape from yellow piles
  rt random 360
  fd 20
  if pcolor != black
    [ get-away ]
end
```

方向をランダムに0～360度右回転し
20進んで
パッチ色が黒でなければ、離れる(get-away)

※get-awayが再帰的になっていることに注意

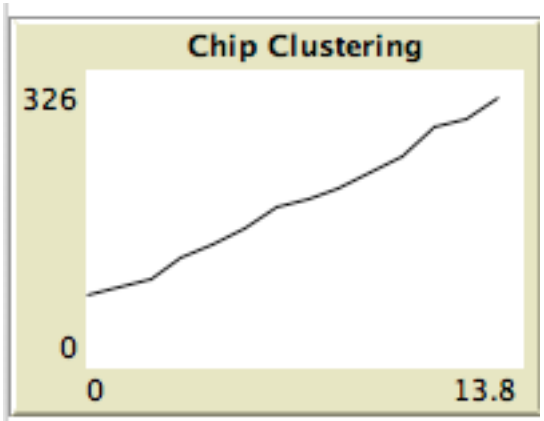
Termits : 小刻みに動く　wiggle

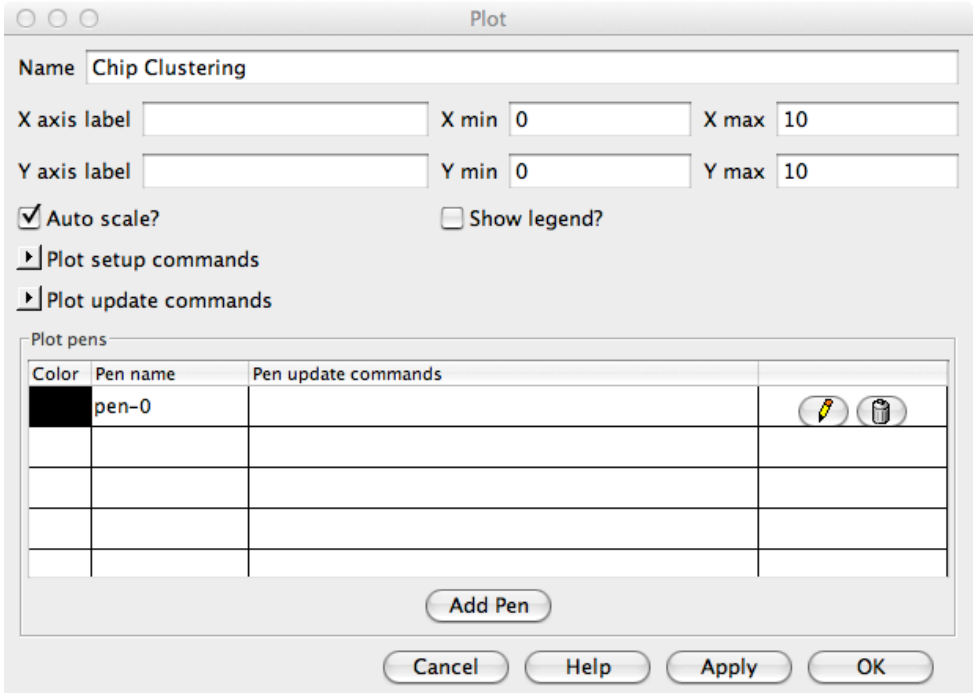
```
to wiggle ; turtle procedure
  fd 1
  rt random 50
  lt random 50
end
```

1進んで
ランダムに0～50度右回転して
ランダムに0～50度左回転する

Termits : プロットの追加

4近傍のパッチ色がすべて黄色のパッチ数を時系列でプロット

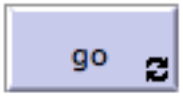





```
to draw-plot
  set-current-plot "Chip Clustering"
  plot count patches with
    [                                    ]
end
```

Termits : goの変更

goコマンドボタンをobserverにして、goプロシージャ内にaskを追加




```
to go  ;; turtle procedure
  ask turtles [
    search-for-chip
    find-new-pile
    put-down-chip
  ]
  draw-plot
end
```