スパイロラテラルとタートルグラフィックス (2)
パターソンの虫

SHIMURA Masato
jcd02773@nifty.ne.jp

2011 年 6 月 24 日

目次

概要

スパイロラテラルの第 2 弾としてコンウエイが考案しパターソンが理想化した葉脈を捕食する「パターソンの虫」の四角虫と 3 角虫を作成する。食欲旺盛な 3 角虫は幾何デザインとしても優れている。

1 格子を喰らう虫

本稿は「スパイロラテラルとタートルグラフィックス (1)」と同様、マーチン · ガードナーの「プログラムされた虫」に拠っている。

かのコンウエイが虫を探求者ではなく捕食者とし、ある大きさの格子状の葉脈を捕食する方法を示した。

1971 年にウオーリック大学の計算機科学者 *Michael Paterson* が「パターソンの虫」としてこの大昔の虫を数学的に理想化した

ガードナーは *MIT* 人口知能研究室から刊行された *Michael Biller* の論文「パターソンの虫」を元にしており、多くの知見はビーラーによる。

- 虫はある格子点で卵から孵って動き始め、格子線に沿って線を食べながら這っていく
- 各格子点での虫の動きは統一ルールを定めておく
- 一度通った道は既に食べられているので再度通ることは出来ない。
- 一度にまっすぐ進む距離は規定しない。
- 各格子点に到着した時の曲がるルールは決めておく（各格子点の食べられたかまだ食べられていないかのパターンで統一ルールに従う）

1.1 タートルの種類

今までは *fd,rt* を用いてきたが、タートルと座標の関係は次の *3* 種類がある。

船を操縦する場合、進行方向 *NNW* という場合と取り舵 *90* 度と言う場合があり、道を歩く場合でも近所では方位よりもあそこの角を右に曲がって、*100* メーター行って左折と言うように *2* つの座標を併用している。ローカル座標とは距離は尺取虫で方向は進行方向の前後左右の行動を示す座標で、ワールド座標との関係はジャイロコンパスで補正する

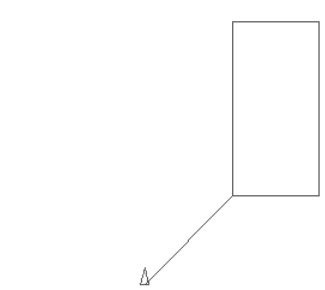
以下は *4* 角虫をモデルに 3 角虫を視野に入れた試行錯誤の結果である。面倒はなるべく避けるという考えで統一した。

	歩幅	角度		
0	ローカル	ローカル	*fd*, *bk*	*rt*, *lt*
1	ローカル	ワールド	*plusx*, *plusy*	*plusxy*
2	ワールド	ワールド	*setx*, *sety*	*setxy*

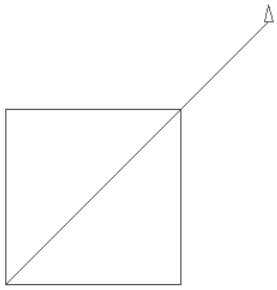
1.1.1 虫の動きの記法

海亀や鮭は太平洋を広範に回遊して産卵に故郷へ帰る。海亀の日々の動きはローカル座標であろうが、亀の頭の中でワールド座標に変換しているかもしれない。

タートルグラフィックスにもワールド座標の関数がある。



setx sety



plusx plusy

setxy　ワールド座標を直接指定

```
show setx 40 sety 80 setx 0 sety 0 setxy _40 _40
```

plusxy　進行はステップを刻むが方位はワールド座標

```
show plusx 4 plusy 4 plusx _4 plusy _4 plusx 6 6
```

四角虫は *plusx plusy* を用いて 1 ステップずつ移動し、角度は用いていない。*3* 角虫は格子が *60* 度 *6* 分割となり正方格子とならないのでいろいろな方法を試行した。

虫の一生　.

虫の誕生　任意の点。

最初の食　*N* 向き　↑*plusy 1* と設定している。この設定を格子に食べたと書き込むか初期値の単なる属性と見るかによって展開に差異が出る。本稿は属性とした。

ジャイロ　　この虫の挙動をワールド座標の格子に書き戻すにはジャイロが有効であるがなかなかに複雑である。虫は賢い。

虫の死　*(*虫は飛べない*)*

- 到着ポイントから派生する格子が全て食べられ、移動できないとき。
- エッジに到着したときは吸着され離脱できないことがあるので回転命令を多用する

show　グラフィックは *show* を用いる

虫といえば *Grace Hopper* 女史を思い浮かべる。女史は *COBOL* の開発者であり、女史に捕獲されて紙にピン留めされたリレーに進入して焦げた最初の虫はスミソニアン科学博物館に永久保存されている。こちらの虫とも大いに格闘した

1.2 格子の構築

将棋は 9×9 の枡で *(1,1)* は右上。囲碁は 19×19 の格子点で *(1,1)* は左上である。捕食される格子は縦横を表示する必要があり単純な *0,1* のマトリクスでは表すことが出来ない。

将棋型	← ← ↓ · ↑ · ↑ → →	ブロック形式は縦横の線がオーバーラップする。 無限少の立体格子は粒子として扱える
囲碁型	↑ ↑ ← · → · → ↓ ↓	クロスを積み上げると縦横の格子が各々重なる
十字を分割	→ → ↓ ⇓ ↓ ⇒ ⇒ ↓ ⇓ ↓	次のような方法がある • 縦と横の *2* 枚のプレートに分けてバイナリーであらわす（三角虫では *4* 枚） • 一枚のプレートで表す場合はボックスで縦　横を *2* ビットのバイナリで表す 0 1 | 1 0 0 0 | 1 1 *2* ビットを *10* 進に変換すると一つの数字になる 1 | 2 0 | 3 下の方法を用いて一つの数字で表現する。

1.3　J の文法（1）2 進法と 10 進法

3 ビットのバイナリー

```
  #: i.8
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
```

3 ビットの 2 進を *10* 進に戻す

```
#.  #: i.8
0 1 2 3 4 5 6 7
```

2　単純 4 角虫

2.1　格子の設計

2.1.1　十字を分割する方法

全体　N ↑ / W ← → E / ↓ S	**反時計回り**にすると左右が正順となる
次のように上と左を各々借りて十字を合成する	$(1,3)$ ↑ / $(2,2)$ ← $(2,3)$ → $(2,3)$ / ↓ $(2,3)$
N　↑	上の *N(1,3)* の第 0 ビットを用いる
W　←	左の *E(2,2)* の第 *1* ビットを用いる
S → / E ↓	*(2,3)* の第 *0* ビット, 第 *1* ビットを用いて *2* 進法で考察し格子には *10* 進化して記入する S E | 10 0 0 | 0 0 1 | 1 1 0 | 2 1 1 | 3

2.1.2　グリッドのスカーフ

行き止まりを表すために指定した個数のグリッドに次のような細工をする

付加　上段に一列, 左に一列付加すると見た目が *1* オリジンとなる
書き換え　最下段の列と右端の列 を *3,2* に書き換え

```
mk_pred_grid_sub 5
3 3 3 3 3 3
3 0 0 0 0 1
3 0 0 0 0 1
3 0 0 0 0 1
3 0 0 0 0 1
3 2 2 2 2 3
```

0 ビット　1 ビット		
↓	→	
S	E	10 進
0	0	0
0	1	1
1	0	2
1	1	3

練習と試行錯誤のためルールが単純な四角虫から作ってみよう。四角虫は生殖力が弱くあまり展開は期待できない。

2.1.3　格子の食前食後の情報

０をまだ食べていない箇所、１を既に食べてしまった状態とする。同じパス（食べてしまった箇所）は *2* 度は通れない

ポジション *(2 3)* の虫の次のステップの十文字の情報は *NWSE* の 4 ポイント（内一つは *approach*）である。

各ポイントは次のように情報を格納する。*S E* の食べたか否を *(0 1)* のを二進法で構成し、*10* 進法に変換すると格子の状態が表現できる。

２進で *(0)* はまだ食べていない線分、*(1)* は食べてしまった線分を表す

現在のポイントを *(2 3)* とすると、*(2 3)* のポイントで *(S E)* が得られる。*N* は *(1 3)* の *S(*第 *0* ビット）から、*W* は *(2 2)* の *E(*第 *1* ビット）から得られる。この４ポイントを合成すると逆時計回りに *NWSE* の状況が得られる。

N W S E	*N*	*W*	*S E*
N 1 3 ↑ *W* 2 2 ← 2 3 → 2 4 *E* ↓ 3 3 *S*	*N* (1, 3) ↓ (0*bit*)	*W* 2 2 → (1*bit*)	(2, 3) → (1*bit*) *E* ↓ (0*bit*) *S*

NWSE の組み合わせは $2^4 = 16$ 通りとなるが、ある格子点に至るにはどこかのパスを通っているのでそこを 3 番目 *(Approach)* の位置を *1* で固定すると変数が一個減って $2^3 = 8$ 通りに単純化でき、ルールの適用が単純になる

選択肢	*N* *Top*	*W* *Left*	*S* *Appr*	*E* *Right*	10 進
3	0	0	1	0	2
2	0	0	1	1	3
	0	1	1	0	6
	1	0	1	0	10
1	0	1	1	1	7
	1	0	1	1	11
	1	1	1	0	14
0	1	1	1	1	15

2.2 ワールド座標とローカル座標　捕食情報データの取り出し

次の *3* 項目を整理しておこう。*position* は格子点で、*Direction* ⇔ *Approach* であるワールド座標とローカル座標の橋渡しは *0123* で行う

項目	ワールド座標	ローカル座標
Direction	*NWSE*	*HLxR*
Position	格子のクロス点	同じ
Approach	*NWSE*	*S* を 1 に固定

1. 格子 *(LATTICE)* からの取り出しと書き込みはワールド座標を用いる
2. *(1)* の虫の位置と捕食情報はワールド座標からジャイロでローカル座標に変換して解析する
3. 解（進行方向*)* はローカル座標で求めてジャイロでワールド座標に戻す
4. *LATTICE* への書き込みはワールド座標で行う

ワールド座標		ローカル座標
N ↑ W ← → E ↓ S	1 3 N ↑ 2 2 W ← → E 2 3 ↓ S 23 現在位置 *(2,3)* とした場合の解析のための *NWSE* データの格納箇所	0 *Head* ↑ 1 *Left* ← → *Right* 3 ⇑ *Approach* 2 虫は尺取を繰り返し常に *Approach* の位置にいる 1 *Left* ↑ 2 *Approach* ⇒ → *Head* 0 ↓ *Right* 3

ワールド座標とローカル座標の関係は煩瑣であるので名刺やカード、*Print* 出来る *CD* に書き込み、回転して確認した。

2.2.1 ワールド座標からローカル座標への変換テーブル

- 格子 *(LATTICE)* は縦横の *2* つの情報を *10* 進に変換した一つの数字で持っている。ワールド座標で取り出したときの *(2,3)* の格子点は次の形をしている

0	1	2	3
N	W	S	E
(1, 3)	(2, 2)	(2, 3)	(2, 3)
0(0*bit*)	1(1*bit*)	2(0*bit*)	2(1*bit*)

- *(1,3)(2,2)(2,3)* を *0,1,2* 番目として次の指標で取り出すとローカル座標は右のようになる

N	→	S	2	2	0	1
W	→	E	2	0	1	2
S	→	N	0	1	2	2
E	→	W	1	2	2	0

0	1	2	3
↑	←	⇑	→
Head	*Left*	*Approach*	*Right*
(0*bit*)	(1*bit*)	(0*bit*)	(1*bit*)

- 回転で作成する

```
 ;("1),. ({@> 2 3 0 1) |.(L:0)  0 1 2 2
2 2 0 1
2 0 1 2
0 1 2 2
1 2 2 0
```

- １０進から２進への変換

NWSE のワールド座標の *4* 個の *10* 進の数字は *2* 進に戻すと次のように各格子点の十文字の情報を表す

```
   #: 2 0 1 3
0 1 (bit)
------
1 0  NB. head
0 0  NB. left
0 1  NB. approach
1 1  NB. right
```

J の `Antibase (#:)` は *0 1* のみの場合は *1* 列しか打ち出さないので２列になるように補わなければならない。

- *0* ビット目が列、１ビット目が行の情報を表す（０はまだ食べられていない。１は既に食べられた）
- ローカル座標でのバイナリーの *0bit,1bit* の取り出し箇所 *(*太字*)*

（*2*）は虫の *approach(*現在位置*)* であり、後で１に固定する *(*のでどちらでも良い*)*

	N,S			*W,E*	
	列	行		列	行
0	**1**	0	0	1	**0**
1	0	**0**	1	**0**	0
2	0	1	2	0	1
3	1	**1**	3	**1**	1

- ジャイロに従って *HLxR* の４ビットのデータを取り出すとローカル座標になりルールが適用できる

2 進に *10* 進を併用して解析する。`#:`と`#.` を用いて変換する

```
   #: 2 0 1 3
1 0
0 0
0 1
1 1
```

approach と取り出すビットの構成。*direct* でなく *approach* を用いている。紛らわしいので *CD-ROM* に書き込んだ図を回して何回も試み、確定した

```
;("1),. ({@>i.4)|.(L:0)  0 1 0 1
0 1 0 1  NB. 0=N  --> S(0),E(1),N(0),W(1)
1 0 1 0  NB. 1=W  --> E(0),N(1),W(0),S(1)
0 1 0 1  NB. 2=S  --> same world
1 0 1 0  NB. 3=E  --> W(0),(S(1),E(0),N(1)
```

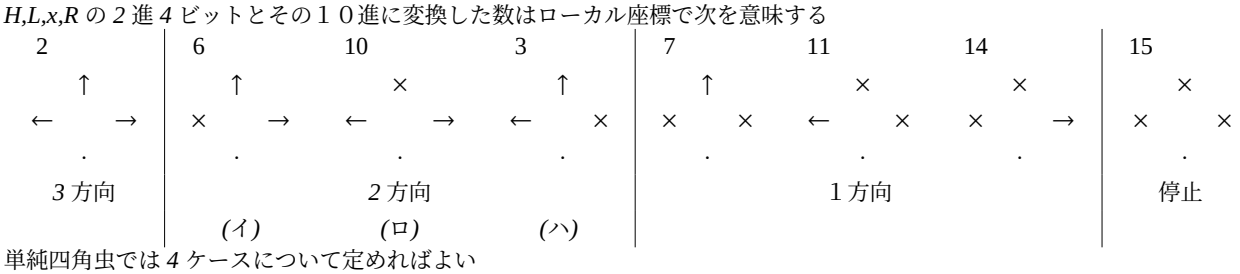
0,1 ビットの交互取り出しにはランク *0 1* を用いる。

```
   i.4 2
0 1
2 3
4 5
6 7


   ({@> 0 1 0 1){"0 1  i. 4 2
0 3 4 7
```

2.3 ルール

2.3.1 ルールのアルゴリズム

H,L,x,R の *2* 進 *4* ビットとその１０進に変換した数はローカル座標で次を意味する

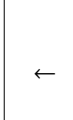
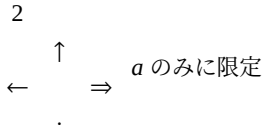
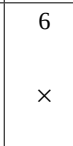
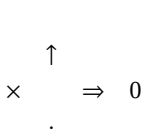
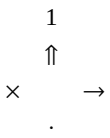
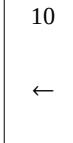
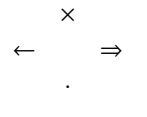
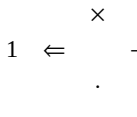
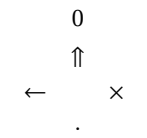
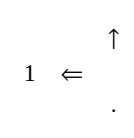


単純四角虫では *4* ケースについて定めればよい

2.3.2 ルールの入力

3 方向は右に固定し、*2* 方向 *(イ)*（ロ）*(ハ)* は 0/1 の *3* 個のベクトルで入力する *(1* を多めに*,1 1 0* など）

まっすぐ進むとエッジに到達して死んでしまい、曲がらないと幾何学模様にならないので曲がり優先のルールである

選択肢	図	*a=0*	*b=1*
3		 *a* のみに限定	
2(イ)			
2(ロ)			
2(ハ)			

2.4 捕食情報の格子への書き込み

2.4.1 解のローカル座標をワールド座標に変換

- ルールに従い虫が次に進む方向が決まる。この方向はローカル座標上のものである
- 方向をワールド座標に変換する

approach → *Direct*	ローカル座標	ワールド座標	変換リスト
$S \to N$	0 ↑ 1 ← → 3 ⇑ 2		
$E \to W$	3 ↑ 0 ← ⇐ 2 ↓ 1	0 ↑ 1 ← ⇐ 3 ↓ 2 時計回りに *90* 度	$0 \to 1 \mid 1$ $1 \to 2 \mid 1$ $2 \to 3 \mid 1$ $3 \to 0 \mid -3$
$N \to S$	2 ⇓ 3 ← → 1 ↓ 0	0 ⇓ 1 ← → 3 ↓ 2 上下を逆にする	$0 \to 2 \mid 2$ $1 \to 3 \mid 2$ $2 \to 0 \mid -2$ $3 \to 1 \mid -2$
$W \to E$	1 ↑ 2 ⇒ → 0 ↓ 3	0 ↑ 1 ⇒ → 3 ↓ 2 反時計回りに *90* 度	$0 \to 3 \mid 3$ $1 \to 0 \mid -1$ $2 \to 1 \mid -1$ $3 \to 2 \mid -1$

2.4.2 格子への書き戻し

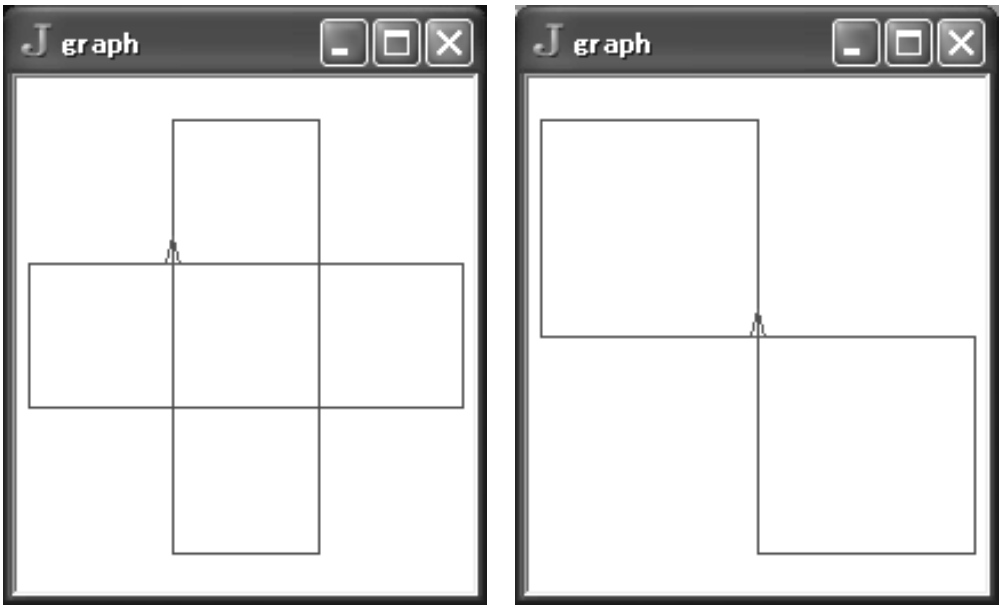
格子は *2* 進に変換すると列が左、行が右に配置されている。0がまだ食べられていない箇所、1が既に食べられた箇所である。現在位置の虫が次のように進む場合の格子の書き換え手順は次の通り

解 *(*ワールド*)*	方向	書き換え箇所	*2* 進	*10* 進	
0	*N*	*(1.3)* の *0(*左*)* ビット	$0\ 0 \to 1\ 0$ $0\ 1 \to 1\ 1$	$0 \to 2$ $1 \to 3$	*+2*
1	*W*	*(2.2)* の *(1)* 右ビット	$0\ 0 \to 0\ 1$ $1\ 0 \to 1\ 1$	$0 \to 1$ $2 \to 3$	*+1*
2	*S*	*(2.3)* の *(0)* 左ビット	$0\ 0 \to 1\ 0$ $0\ 1 \to 1\ 1$	$0 \to 2$ $1 \to 3$	*+2*
3	*E*	*(2.2)* の *(1)* 右ビット	$0\ 0 \to 0\ 1$ $1\ 0 \to 1\ 1$	$0 \to 1$ $2 \to 3$	*+1*

2.5 show

- カンバスは *show* を用いるので大きさを気にすることは無いし、予め大きさを定める必要も無い
- 進む距離は一歩ずつ
- 方向はワールド座標の解を次に変換する

plusy 1
↑
plusx −1 ← → *plusx* 1
↓
plusy −1



2.6 実行のプロセス

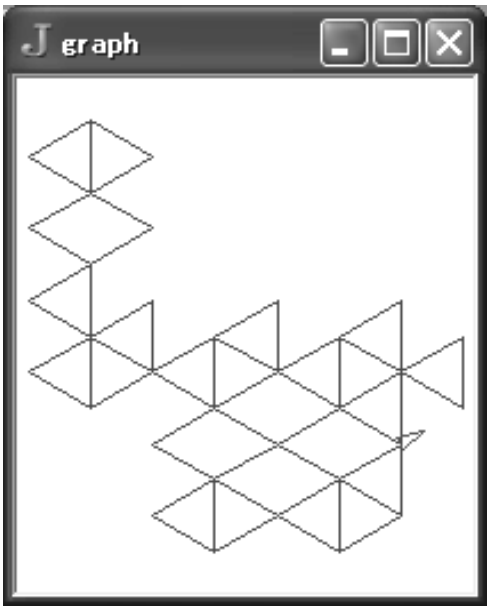
ローカル座標は左右前後の座標、ワールド座標は *NWSE* の座標

processDirect	*Pickup&set*· ローカル座標	解 · ローカル座標	変換· ワールド座標
0 *S* → *N*	0 ↑ 1 ← → 3 ⇑ 2 *(2,3)*	0 ↑ 1 ← ⇒ 3 ⇑ 2 *3 (2,4)*	
1 *W* → *E*	1 ↑ 2 ⇒ → 0 ↓ 3	1 ↑ 2 ⇒ → 0 ⇓ 3	0 ↑ 1 ⇒ → 3 ⇓ 2 反時計回りに *90* 度

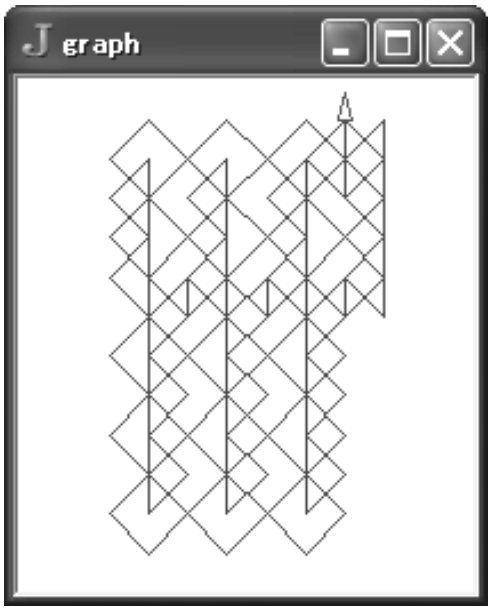
		(2,4)	*3*	*2 (3,4)*
2	$W \to E$	2 ⇓ 3 ← → 1 ↓ 0	2 ⇓ 3 ⇐ → 1 ↓ 0	0 ⇓ 1 ⇐ → 3 ↓ 2
	(3,4)	*3*	*1 (3,3)*	
3	$E \to W$	3 ↑ 0 ← ⇐ 2 ↓ 1	3 ⇑ 0 ← ⇐ 2 ↓ 1	0 ⇑ 1 ← ⇐ 3 ↓ 2
	(3,3)		*3*	*0*

3　単純 3 角虫

単純三角虫は食欲旺盛でルールを適当に与えると食べ尽くした後に美しい幾何学模様を残す。生命の源泉は壁にあたった後の反転である



(200;10; 5 5) worm3 1; 2; 1 0 2 2 1;1



(200;10; 5 5) worm3 0; 2; 1 1 2 2 1;1

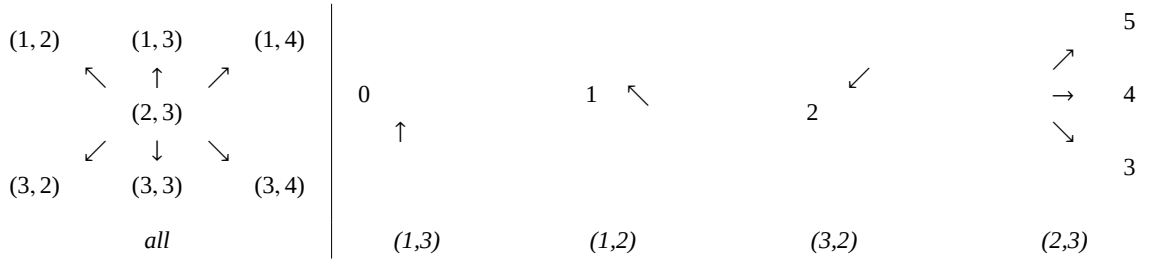
僅か 10 × 10 の格子でもこのように *100* 回前後の反復動作が記録される。ガードナーの記事には *1000* 万回食べつくしてもまだとまらない図形が紹介されている。まだチューニングが足りないようだ。

3.1　格子の設計

格子はユニオンジャックから縦又は横の線を除いた *60* 度 *6* 分割の正三角形を用いるには横の線を除く方法と縦の線を除く方法があり、座標が少し異なる。今回は *NS* を用いて、*WE* を除いた。

```
1     0     5    WNW     N     ENE
   ↖  ↑  ↗            ↖  ↑  ↗

   ↙  ↓  ↘            ↙  ↓  ↘
2     3     4    WSW     S     ESE
```

↑↖↙ は各々 *(1,3),(1,2),(3,2)* から *1* ビットづつ借用する。↓↘,↗ は *(2,3)* の *3* ビットを用いる



合成すると 6 ビットになる。このうち 3 を *approach* に用いる。

6 **角形と** show　格子を正三角形で 6 分割する。*show* に渡すデータは次の *3* 通りを考える

1. *plusxy* タートルグラフィックスの *plusxy* は角度を計算しなくとも斜線を正しく出力する。これは *45* 度線である。正六角形の方がデザイン的には綺麗だが正方格子にはならない。
   格子の役割は捕食経過の記録であり、マトリクスを用いた正方格子で次により情報を記述する。

1	0	5
plusxy $-1, 1$	*plusy*1	*plusxy* $1, 1$
plusxy $-1, -1$	*plusy* -1	*plusxy* $1, -1$
2	3	4

2. *cos,sin* で歪んだ格子をグラフィックスにする。

```
dfr=: *&(180%1p1) NB. degree from radian
rfd=: *&(1p1%180) NB. radian from degree
cosa=. %  2&o.  dfr 1
sina=. %  1&o.  dfr 1
```

[*1]

3. ローカル座標の軌跡をそのまま *fd* を用いて記録する。この方法でグラフィックスが綺麗に表現できる。

1	0	5
*lt*60 *fd*1	*fd* 1	*rt* 60 *fd* 1
lt 120 *fd* 1	*fd* -1	*rt* 120 *fd* 1
2	3	4

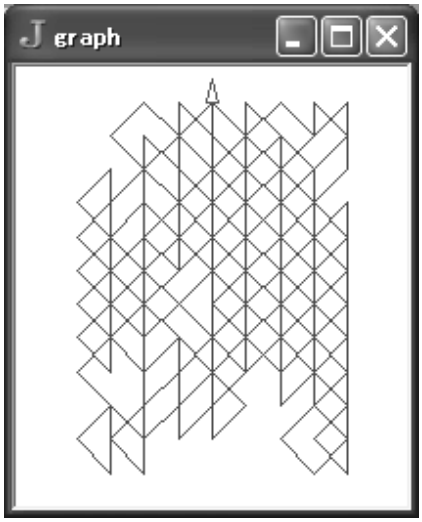
3.2　J の文法 (2)

3 **本足の** 2 **進と** 10 **進**　　*(2,3)* の箇所は *2* 進 *3* ビットで構成され、*10* 進に変換後一つの数字で格子に格納される

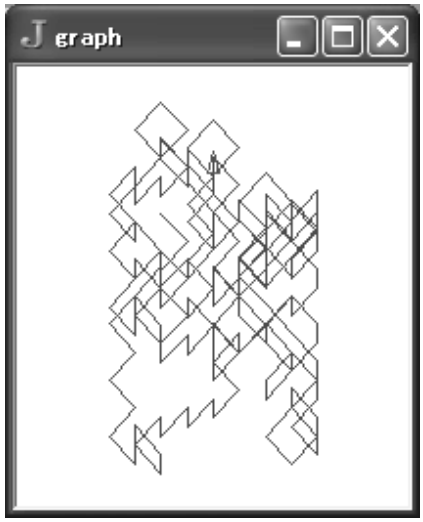
```
(a=. #: i.8),.i.8
```

選択肢	0	1	2	10 進
3	0	0	0	0
2	0	0	1	1
	0	1	0	2
	1	0	0	4
1	0	1	1	3
	1	0	1	5
	1	1	0	6
0	1	1	1	7

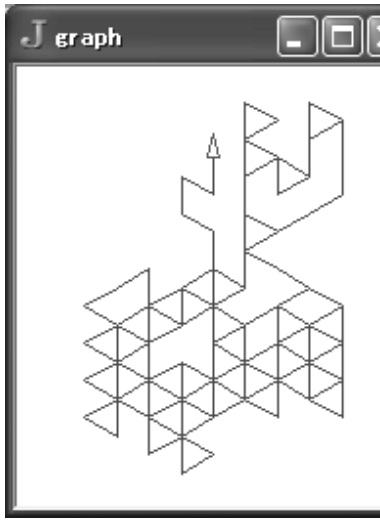
[*1] 1p1 is π



(200;10; 5 5) worm3w0 0; 1; 2 2 1 2 0;1 1



(200;10; 5 5) worm3w1 0; 1; 2 2 1 2 0;1 2



(200;10; 5 5) worm3w 0; 1; 2 2 1 2 0 3

show は *3* 通り用意した。

捕食と進路選択-ルールの前提　進路選択で５は一通り、４は *5* 通り、*3* と *2* は各々 *10* 通り、１は *5* 通りあり、死滅は *1* 通りである。*S* はローカル座標では *approach* で、常に *1* となる。

```
(a=. #:i.32)
 a1=. (3{."1 a),.1,. _2{."1 a
   /:~ (+/"1 a1),.  a1,.#. a1
```

選択肢	*N*	*WNW*	*WSW*	*S*	*ESE*	*ENE*	10 進
5	0	0	0	1	0	0	4
4	0	0	0	1	0	1	5
	0	0	0	1	1	0	6
	0	0	1	1	0	0	12
	0	1	0	1	0	0	20
	1	0	0	1	0	0	36
3	0	0	0	1	1	1	7
	0	0	1	1	0	1	13
	0	0	1	1	1	0	14
	0	1	0	1	0	1	21
	0	1	0	1	1	0	22
	0	1	1	1	0	0	28
	1	0	0	1	0	1	37
	1	0	0	1	1	0	38
	1	0	1	1	0	0	44
	1	1	0	1	0	0	52

選択肢	*N*	*WNW*	*WSW*	*S*	*ESE*	*ENE*	10 進
2	0	0	1	1	1	1	15
	0	1	0	1	1	1	23
	0	1	1	1	0	1	29
	0	1	1	1	1	0	30
	1	0	0	1	1	1	39
	1	0	1	1	0	1	45
	1	0	1	1	1	0	46
	1	1	0	1	0	1	53
	1	1	0	1	1	0	54
	1	1	1	1	0	0	60
1	0	1	1	1	1	1	31
	1	0	1	1	1	1	47
	1	1	0	1	1	1	55
	1	1	1	1	0	1	61
	1	1	1	1	1	0	62
0	1	1	1	1	1	1	63

4 角虫で思索したアルゴリズムを踏襲し、スクリプトも変更を最小限にする

3.3 格子の設計

格子の縁はオーバーゾーンしないで確実に反射させるためには近寄らせないか、オーバーゾーンしたら *1* 行 *1* 列付加するか、バリアを設けて入れなくする。最期の手法により *2* 重バリアとした。

```
   mk_lattice_3 7
7 7 7 7 7 7 7 7 7 7
7 1 1 1 1 1 1 3 7
7 0 0 0 0 0 0 3 7
7 0 0 0 0 0 0 3 7
7 0 0 0 0 0 0 3 7
7 0 0 0 0 0 0 3 7
7 0 0 0 0 0 0 3 7
7 0 0 0 0 0 0 3 7
7 0 0 0 0 0 0 3 7
7 6 6 6 6 6 6 7 7
7 7 7 7 7 7 7 7 7
```

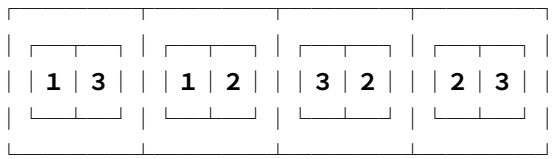
3.4 座標とデータの取り出し

取り出しと書き込みはワールド座標で行う

ワールド座標での取り出し *4* ポイントの座標の取り出しのインデクス

base	0	1	2	3
	N	*WNW*	*WSW*	*S*
(2,3)	(1,3)	(1,2)	(3,2)	(2,3)
±	−1,0	−1,−1	+1,−1	0,0

```
  pick_world_position_3 2 3   NB. index
```



判別のためのデータの構成 取り出した *4* 個のデータを *2* 進に戻し *6* 方位の *2* 進データを構成する。

compose_bin_6pos

進入路 *(approach)* は *local* 座標ではで 3 に固定し、ここは常に *1*（食済み）とする

ローカル座標への変換ージャイロ ワールド座標を虫の座標に変換する。虫の現在位置は常に 3 とし、進行方向の反対 $N \leftrightarrow S$ を 3 にあわせて虫の前方、左右斜め前後ろの 5 ポイントと *approach* を含む 6 ポイントの取り出しのローカル指標（*4* ポイント）を作成する

gyro-local の指標

approach	→	*direction*						
0(*N*)	→	3(*S*)	3	3	3	0	1	2
1(*WNW*)	→	4(*ESE*)	3	3	0	1	2	3
2(*WSW*)	→	5(*ENE*)	3	0	1	2	3	3
3(*S*)	→	0(*N*)	0	1	2	3	3	3
4(*ESE*)	→	0(*WNW*)	1	2	3	3	3	0
5(*ENE*)	→	2(*WSW*)	2	3	3	3	0	1

3.4.1 J の文法 (3)

- マトリクスの回転　ランク"0 1 を用いる
  approach から見た *local* 座標での格子からのデータ採り方の指標

```
   3 4 5 0 1 2 |. "(0 1) 0 1 2 3 3 3
3 3 3 0 1 2  NB. 0
3 3 0 1 2 3  NB. 1
3 0 1 2 3 3
0 1 2 3 3 3
1 2 3 3 3 0
2 3 3 3 0 1
```

  approach から見た各方向で採るビット
  取るカラムの指標作成

```
   3 4 5 0 1 2 |. "(0 1) 0 1 2 0 1 2
0 1 2 0 1 2
1 2 0 1 2 0
2 0 1 2 0 1
0 1 2 0 1 2
1 2 0 1 2 0
2 0 1 2 0 1
```

- ビットの取り出し
  0,1,2 ビットの取り出しにもランク (" 0 1) を用いる

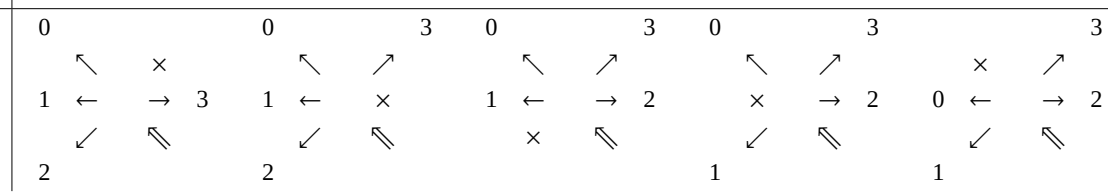
```
   i. 6 3
 0  1  2
 3  4  5
 6  7  8
 9 10 11
12 13 14
15 16 17


   ({@> 0 1 2 0 1 2){ "0 1 i. 6 3
0 4 8 9 13 17
```

3.5 ルール

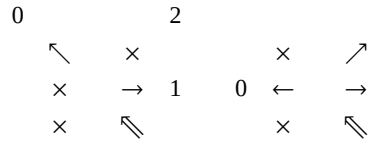
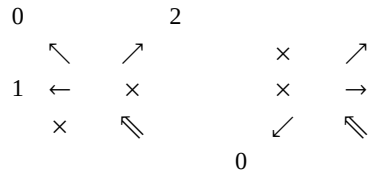
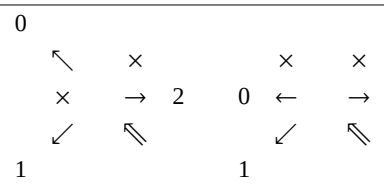
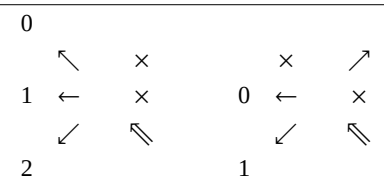
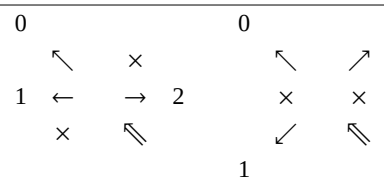
5　　選択肢 5 は一個のみ。直進は × とし、右回り *60* 度と *120* 度の *2* パターンのみとする。左回りは取り入れない。

	4	
	N 1	
5	↖ ↗ ← → 0	*0/1 only*
	↙ ⇖ *S*	

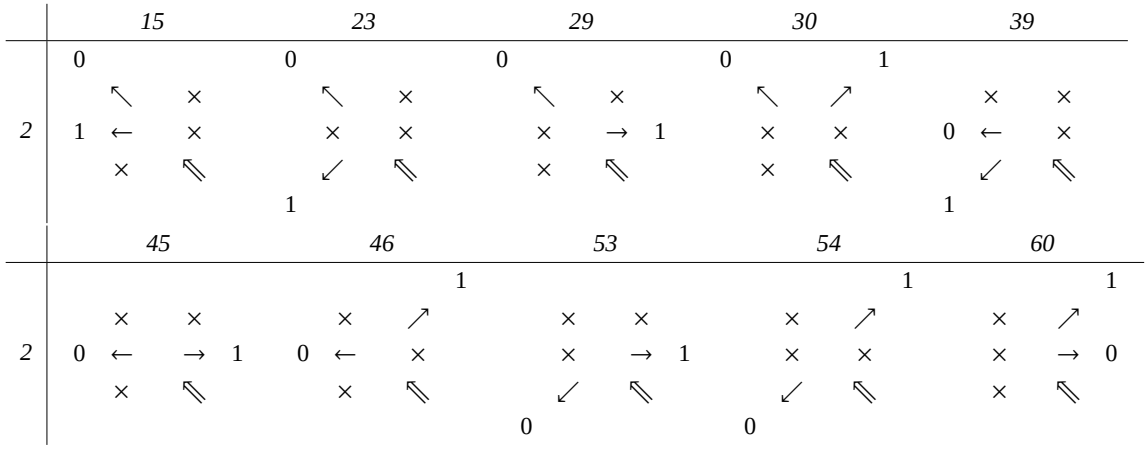
4　　*4* は *5* 個

	5	*6*	*12*	*20*	*36*
4					

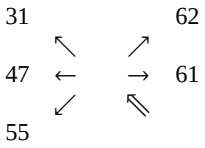
3　振り付けは3で行う。ガードナーは3を4パターンに分けている。これを踏襲し、ガードナーの定義から漏れていた１３と２２を付け加え、*5* 個のベクトルで定義する、

	28	*44*
3（イ）		
	14	*52*
(ロ)		
	21	*37*
（ハ）		
	7	*38*
3（ニ）		
	13	*22*
（ホ）		

2　選択肢２は１０パターンある



1　*1* 方向のみ。番号は *10* 進のケース



この選択の組み合わせは

$2 \times 4 \times 3^5 \times 2 = 3888$ 通りある

ループの下に `1!:2&2 ctr` とループの回数を表示し、*3* 角虫がどれ位生存できたかを表示するようにした。

3.6 戻しのジャイロ

ローカル座標をワールド座標に変換するパラメータ　機軸は *approach*

0				*1*				*2*			
loc	→	*wld*	*diff*	*loc*	→	*wld*	*diff*	*loc*	→	*wld*	*diff*
0	→	3	3	0	→	4	4	0	→	5	5
1	→	4	3	1	→	5	4	1	→	0	−1
2	→	5	3	2	→	0	−2	2	→	1	−1
3	→	0	−3	3	→	1	−2	3	→	2	−1
4	→	1	−3	4	→	2	−2	4	→	3	−1
5	→	2	−3	5	→	3	−2	5	→	4	−1

3				*4*				*5*			
				loc	→	*wld*	*diff*	*loc*	→	*wld*	*diff*
				0	→	1	1	0	→	2	2
				1	→	2	1	1	→	3	2
same world coordinance				2	→	3	1	2	→	4	2
				3	→	4	1	3	→	5	−2
				4	→	5	1	4	→	0	−4
				5	→	0	−5	5	→	1	−4

writeback data **のパラメータ**　*3* ビットの捕食データのうち今回食べた箇所を 1 に変換し、*10* 進に戻して格子に書き込む

0,3									*1,4*									*2,5*								
変	換	前	変	換	後	10	進		変	換	前	変	換	後	10	進		変	換	前	変	換	後	10	進	
0	1	2	0	1	2	前	→	後	0	1	2	0	1	2	前	→	後	0	1	2	0	1	2	前	→	後
0	0	0	1	0	0	0	→	4	0	0	0	0	1	0	0	→	2	0	0	0	0	0	1	0	→	1
0	0	1	1	0	1	1	→	5	0	0	1	0	1	1	1	→	3	0	1	0	0	1	1	2	→	3
0	1	0	1	1	0	2	→	6	1	0	0	1	1	0	4	→	6	1	0	0	1	0	1	4	→	5
0	1	1	1	1	1	3	→	7	1	0	1	1	1	1	5	→	7	1	1	0	1	1	1	6	→	7
+4									*+2*									*+1*								

3.7 J の文法 (4) 指標と取り出し、書き込み

マトリクスからスポットで数値を出し入れする指標は *J* 言語の開発段階でもトロントは最期まで苦労していた。

指標（1） 次のようなポイントを指定したボックスを用いると便利である

```
   1 2;2 3;3 4
+---+---+---+
|1 2|2 3|3 4|
+---+---+---+

   (1 2;2 3;3 4){i. 5 5
7 13 19
```

```
   i.5 5
 0  1  2  3  4
 5  6  7  8  9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24
```

アメンド ··· 書き込み

```
   100 200 300 (1 2;2 3;3 4) } i. 5 5
 0  1   2   3   4
 5  6 100   8   9
10 11  12 200  14
15 16  17  18 300
20 21  22  23  24
```

取り出しとランク ベクトルやマトリクスから取り出した数値は元の形のランクが残っていることがある。特にボックスが介在した場合に良く見受けられる。

```
$ 2{  1 2 3
3


$ 2{  1 2 3
```

この場合は*$*で検査してもランクが残っていないのでスカラである。

ここが *1* となる場合は、これは一個の数で構成するベクトルでランク *1* の属性を持っている。これを上のように書き込もうとするとランクエラーとなる。*ravel items* を用いて `(,./)` として減次してなければならない

簡易デバッグ `1!:2&2` を用いて変数を画面に表示させると経過が追跡できる。

```
 1!:2&2 DIRECT0;DIRECT
```

References

マーチン · ガードナー 一松 信訳「マーチン · ガードナーの数学ゲーム *I(*新装版*)*」日経サイエンス社　*2010*

*SHIMURA Masato [*スパイロラテラルとタートルグラフィックス（１）プログラムされた虫」*JAPLA 2011/05*

`http:`*/japla.sakura/ne.jp/* の *Workshop May 2011*

J602 J701 はトロントから *DL* 出来ます
`http:`*/www.jsoftware.com/*
スクリプトは次から *DL* できます
`http:`*/japla.sakura/ne.jp/* の *Workshop May 2011*