世界モデル実習（SR）・人間の安全保障実験実習Ⅲ（GR）　　2005 年度夏学期

担当　山影進
TA　阪本拓人、鈴木一敏、保城広至、
光辻克馬、山本和也

第 12 回　複雑なエージェントをすっきりと (7 月 12 日)

今日の目的:
エージェントに複雑な処理をさせると、ルールが複雑に、そして読みにくくなってきます。そこで今回は、複雑なルールをまとめて、すっきりさせる方法を学びます。また、エージェントごとの実行順序の細かな指定方法と、同期の取り方についても学びます。

●　複雑なルールをすっきりと書く

ある程度の規模のモデルを作っていると、何度も同じ処理を書かねばならない場面が度々あります。特に、重複する部分が複雑に場合分けされている場合などは、あとで見たときに読みにくいばかりでなく、if や endif の数を間違えるなど致命的なミスを誘発しかねません。そんなときには、サブルーチンやファンクションを用いると、重複部分を一カ所にまとめたうえで簡単に呼び出して使うことができます。

＊ ファンクション(自作の関数を定義する)

戻り値が必要な場合(中で計算した値などを元のルールに戻った後も使いたい場合)には、ファンクションを利用してください。ルール最後尾、Agt_step を「｝」で閉じたあとに、以下のように定義します。

```
Function 関数名(パラメータ宣言)As 関数戻り値の型｛
変数の型宣言部
実行部
Return(変数名)　　← いくつあっても構いません。場合分けごとに特定の値を返すこともできます
}
```

たとえば、前方にいるエージェントの集合を作りたければ、

```
Function countagtsahead(distance as Double) As Integer
{
```

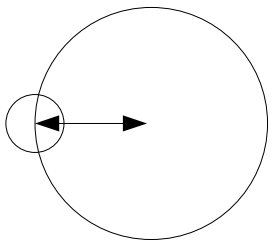
```
dim set as agtset
dim kazu as integer

forward(distance)                                  ← 指定した距離進みます
MakeAllAgtsetAroundOwn(set, distance, False)       ← 集合を作ります
kazu = Countagtset(set)                            ← エージェントを数えます
forward(-distance)                                 ← 元の位置に戻ります
Return(kazu)                                       ← kazu の中のものを返します
}
```

とすると、目の前の丸い範囲内(下図)に居るエージェントをリストアップすることができます。Agt_Init や Agt_Step のルール中では、

```
if countagtsahead(1.2) > 0 then     ← 1.2 前方から半径 1.2 以内に何か居たら
        my.speed = 0                ← スピードをゼロにする
end if
```

などとして、他の関数と同様に使うことができます。相互参照していなければ、作った関数を別の関数の定義で使っても構いません。



＊ サブルーチン(ルールの一部をまとめる)
重複する処理をまとめてしまいたいだけの時は、サブルーチンを使います。

```
Sub 名前(パラメータ宣言) {
変数の型宣言部
実行部
}
```

と定義すれば、ルール内で「名前()」として呼び出せます。パラメータ宣言の()の中は、ルールから受け渡したい値がある時だけ書きましょう。たとえば、「前方の空間が空いていたら進む。進む距離と、どの程度の範囲を見て空いていると判断するかは自分で指定する。」というサブルーチンを定義するとしたら、

```
Sub forwardspace(dist as double, sight as double){

dim set as agtset

forward(dist)                                  ←とりあえず 1 進む
makeallagtsetaroundown(set, sight, false)  ← 進んだ先で周りを見る
if countagtset(set) > 0 then                   ← 何かが居たら
        forward(-dist)                         ← 戻る
end if
}
```

とします。そして Agt_Step で forwardspace(1, 1) と記述すれば、「前方に 1 進み、そこから 1 の範囲内に何も居なければそこに止まる。何か居た場合には元の場所に戻る」という命令が実行されます。サブルーチンには戻り値がありません。したがって、ファンクションの場合と違って「if 処理 1() > 0 then ･･･」のように値の代わりに使うことは出来ません。「空間の縁にいて進めなかった場合は-1 になる」などとしたい場合には、ファンクションを使いましょう。

例題①

50×50(ループなし)の空間を作りましょう。この空間ではモノは右に落ちます。毎ステップ、左下から 1 個ボールが現れ、右側に転がり落ちてゆきます。右に何もなければ端まで毎ステップ 1 ずつ進みます。自分の右に別のボールがある場合、一つ上側が空いていれば、上に 1 ずれます。上記のファンクションを使って、書いてみましょう。Universe で毎回一つずつエージェントを作ります。

```
Agt_Step{
if CountAgtAhead(0.5) == 0 then  ← もし 0.5 先から半径 0.5 以内に何もなければ･･･
        forward(1)               ← 1 進みます
else                             ← もし何か居たら
        turn(90)                 ←上を向きます
        if CountAgtAhead(0.5) == 0 then  ←もし、何もなければ、
```

```
        forward(1)              ← 上に進みます
        end if
        turn(-90)               ← 元の向きに戻ります
end if
}
```

おまけ: Forward()を使って何かを追いかける場合、もしくは、ある目的地へ向かいたいとき、目標への方向を知る必要があります。アークタンジェントを使うのですが、場合わけなど少々面倒な作業が必要です。そんな時に便利な関数をアップロードしておきますので、コピーして使ってみてください。

● 実行順序

＊ エージェントの実行順序

地味に見えますが、実行順序は最も重要な設定の一つです。一見きちんと動いているように見えて、自分が思っているのと別のことが起きている場合もあります。実行順序は、「設定＞実行環境設定＞実行順序」で、細かく設定することができます。現在は、以下の 4 種類の実行順序を指定できます。

・ ランダム

エージェント種に関わらず、全てをシャッフルします。

・ ランダム(初回のみ実行順序を並び替え)

初回のみ、エージェント種に関わらず、全てをシャッフルします。2 ステップ目からは初回と同じ順序で実行します。

・ エージェント種別指定

エージェント種ごとの実行順序を指定できます。同じエージェント種の内部では毎回シャッフルされます。左端の数字が実行順序を表します。

1, オオカミ
2, ヒツジ
2, ヤギ

とすると、オオカミを最初に実行し、その後、ヒツジとヤギはごちゃ混ぜにして実行します。

・エージェント種別指定(初回のみ実行順序を並び替え)
エージェント種ごとの実行順序を指定できます。初回のみ同じエージェント種の内部での実行順序をシャッフルします。

初回のみ実行順序を並び替えるオプションは意外に重要です。たとえば、■が逃げて、▲が追いかけるモデルの場合、

step1	step2	step3	step4
■→▲	▲→■	▲→■	■→▲

という順序では、たとえ▲■のスピードが同じでも、その間の距離が伸びたり縮んだりしてしまいます。

step1	step2	step3	step4
■→▲	■→▲	■→▲	■→▲

と言う、初回の順序を守らなければなりません。

＊　実行順序調整のテクニック
・フラグ
相手の反応を見て対応するような複雑なモデルになると、一通りの動作を 1 ステップだけで書ききれないこともあります。一通りの動作に 3 ステップかかるとすると、3 ステップで 1 ターンとして、それを繰り返す必要があります。こんなときにはフラグを使います。universe直下にフェーズを表す変数を作り、0→1→2→0→1→2→0 という風に変化するようにルールを書きます。

```
universe.phase = universe.phase + 1
if universe.phase == 3 then
        universe.phase = 0
end if
```

(もしくは、universe.phase = getcountstep() mod 3)
そして、エージェントルールでは、

```
if universe.phase == 0 then
        フェーズ 1 のルール
```

```
elseif universe.phase == 1 then
        フェーズ 2 のルール
else
        フェーズ 3 のルール
end if
```

と書きます。各フェーズのルールをサブルーチンにしてしまえば、よりすっきりとわかりやすくなるでしょう。

・空回し

Gethistory()で過去 10 回分の記憶を用いて判断する場合、11 ステップ未満では存在しない過去の記憶を参照してしまう、という問題がよく起きます。これも上記のようにフラグを用いてステップを空回しすることで対処できます。11 特定ステップ未満では別のルールを実行するようにすれば良いわけです。同じ方法で、初期配置を行ったり、途中で状況を変えることが出来ます。

課題

① おまけの関数を使って、2 匹のエージェントが追いかけっこをするモデルを作ってください。空間はループさせると良いでしょう。追いかけるエージェントは自動的に追いかけます。逃げるエージェントはランダムな場所からランダムな方向に一直線に逃げます。スピードは、両方 1 としておきましょう。 (実行環境設定で、ガベージコレクションを 1 としてください。スピードが速すぎる場合は、実行ウェイトで調整してください。Getagt か For each を使います。実行順序に気を付けましょう。逃げる方が先です。)

② 逃げるエージェントの方向を、コントロールパネルで操作できるようにしてみてください。同じ場所からスタートする様にしてみましょう。追う方は、10 ステップ待ってからスタートします。

③ 下図のように、ある一定の視角内の前方だけを見る関数を作ってみてください。