国際政治と情報（2005 年後期）

担当　田中明彦 tanaka@ioc.u-tokyo.ac.jp
TA　　阪本拓人 takutos@nifty.com
保城広至 hoshiro@ioc.u-tokyo.ac.jp

第７回　続・エージェント、変化する！（11 月 25 日）

●概略

○ 前回の宿題の解答例と解説
○ 高度なエージェントの生成・配置方法：For 文のネスト
○ 過去の状態に依存するエージェントの変化：GetHistory 関数と「同期」の問題
○ 宿題

●前回の宿題の解答例と解説

[５] 前回学んだコントロールパネルを用いて、「風邪引き」モデルの Hito の数（今回作ったモデルでは 200 に固定されていました）、シミュレーション開始時の「風邪引き」エージェントの割合（同 0.03）、および風邪の回復までにかかる「感染期間」（同 10 ステップ）を操作できるようにしてみましょう。その上で、各々関心のあるパラメータを動かしてみて、風邪の蔓延の仕方がどのように変化するのかを確かめてみてください。

＞特に取り上げて解説する事項はありません。コントロールパネル作成の手順に従って、Universe 直下に３つの変数を作成し（変数の型に注意）、ルールエディタ中の数値部分をこれらの変数に置き換えるだけです。

[６] 「風邪引き」モデルを修正して、Hito が一度風邪から快復すれば、風邪への耐性を身に付ける、つまり風邪に一切感染しなくなる状態を新たに追加してみてください。この「無敵」状態の表示色は黄色にしましょう。風邪の蔓延の仕方にどのような変化が生じるでしょうか。

＞「健康」「風邪引き」に加えて第三の状態が出てくるので、ルールが複雑になると思われるかもしれませんが、実は Agt_Step のルールを一カ所だけ書き換えれば終わりです。具体的には、「感染期間」完了時のルールを下記のように変更します。なぜこれでいいのかはエージェントのルール全体を見渡せば明らかになるはずです。

```
// 風邪を引いている場合は感染期間をチェック
If My.State == 1 Then

        // 感染ステップをカウント
        My.Counter = My.Counter + 1

        // 10 ステップ過ぎたら風邪に対して無敵になる
        If My.Counter == 10 Then
                My.State = 2
                My.Color = Color_Yellow
        End If

End If
```

この部分だけを書き換えれば OK

[7]「風邪引き」モデルを修正して、Hito が周囲の Hito から無作為に一人「話し相手」を選び、この相手が風邪を引いていれば、自分も風邪を引くというふうにルールを変更してみましょう。GetAgt 関数を用います。

＞前回資料 16 ページの解説を読んで理解していればそれほど難しくないはずです。エージェント集合型変数に集めた周囲のエージェントのうち一人を、GetAgt 関数を用いてランダムに取り出し、風邪引きかどうかをチェックすることになります。エージェント集合が空集合の場合、GetAgt 関数はエラーを起こすので、集合の要素の数を数えるのも忘れないようにしてください。具体的には、エージェント集合をカウントするための整数型変数（下では n）を宣言した上で、For Each 文の部分のルールを下記のように変更します。

```
// 隣接している他の Hito を集めてカウント
MakeOneAgtSetAroundOwn( rinjin, 1, Universe.Field.Hito, False )
n = CountAgtSet( rinjin )

// 周囲に Hito がいれば隣人一人をランダムに選んで接触
If n > 0 Then
        aite = GetAgt( rinjin, Int( n * Rnd( ) ) )

        // もし自分が健康で aite が風邪を引いていたら風邪をもらう
        If My.State == 0 AND aite.State == 1 Then
                My.State = 1
                My.Color = Color_Red
        End If

End If
```

空集合でないかチェック

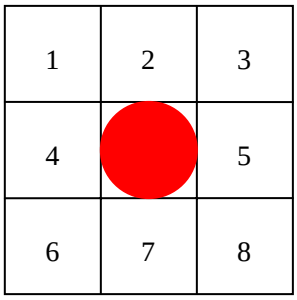
ランダムに一人を選別

●本日のモデルとそのポイント

今回は、マルチエージェント・シミュレーションの世界ではとてもよく知られた、「ライフゲーム」と呼ばれるモデルに挑戦してみます。いくぶん抽象的なモデルですので、最初にこのモデルの背景を簡単に説明しておきます。

ライフゲームとは、ジョン・フォン・ノイマンという数学者が考案したセル・オートマトンをイギリスの数学者ジョン・H・コンウェイが、単純にしたものです。

セル・オートマトンとは、格子の空間において、格子内の各セルが周囲のセルの状態に依存して、自らの次の状態を決めていくという変化が無限に行われるしくみのことです（下図参照）。



赤マルの状態は、周囲の 8 つのセルの状態で

次にどのような状態になるかが決まる

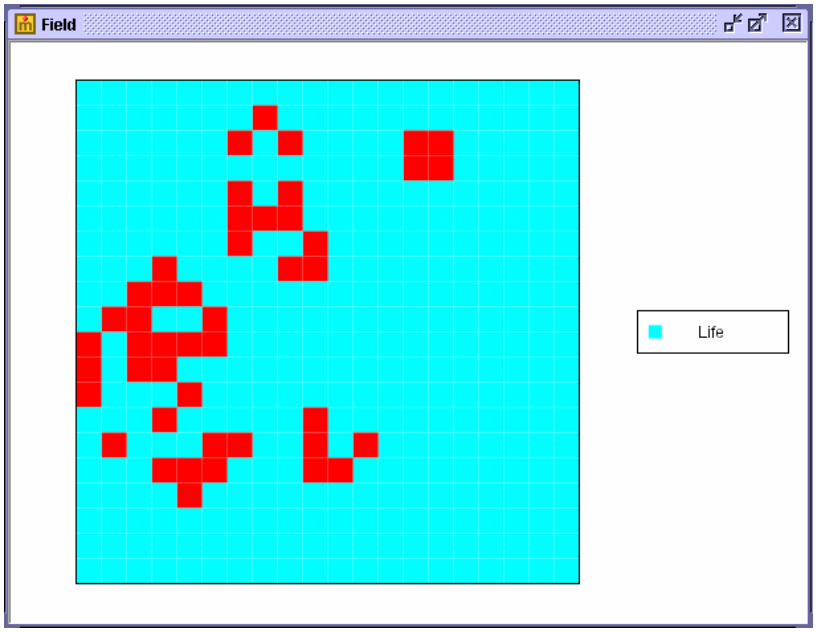
フォン・ノイマンは、20 万個のセルと 29 パターンの状態をとるセル・オートマトンを考えました。天才のフォン・ノイマンにとっては、こんな複雑なモデルも単純だったかもしれませんが、普通の数学的能力の人にとっては、これは複雑すぎます。そこで、（同じく天才的なのですが）コンウェイは、フォン・ノイマンのモデルを単純化して、凡人でも何とか面白いと思えるモデルを作りました。これが、ライフゲームです。

ライフゲームでは、各セルは「生」と「死」２種類の状態しか取りません。セルの状態変化のルールも、次のようにいたってシンプルなものです。

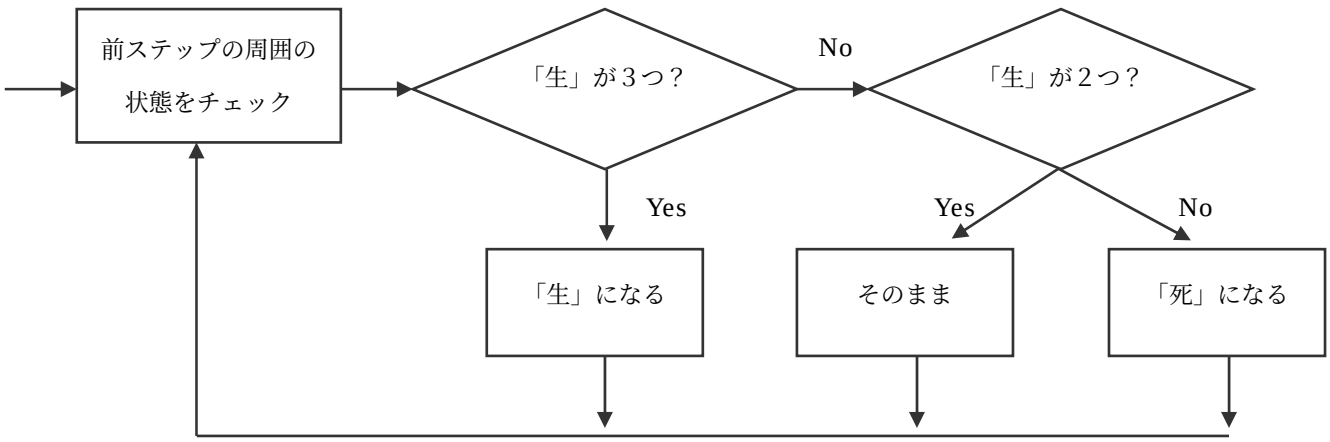
1. 周囲が 3 つ「生」なら、中央は「生」になる。
2. 周囲が 2 つ「生」なら、中央の状態は変化しない。
3. それ以外では、中央は「死」になる。

二次元空間のなかに、この単純なルールを持つエージェントを配置すると、わずかな初期状態の違いで、さまざまな状態変化のパターンが現れます（次ページ図参照）。こんなライフゲームにいったいどんな意味があるのかと思う人もいるかもしれません。しかし、実は、この単純で抽象的な発想は、現代科学の最先端の関心になっているのです。

具体的にいえば、その様子は、空間に生き物がいるようにみえるので、人工生命（AL）研究で関心を集めています。また、わずかな初期状態の相違がまったく異なる状態をもたらすという点で、カオス研究の関心にもなっています。学問的なところから少し距離を置いても、自分でさまざまな初期状態を配置して、面白い状態を生み出すことを楽しむことができるということで、娯楽としての愛好家もいます。実際、発案者のコンウェイは、天才数学者でありながら、この娯楽に熱中したというエピソードを持った人物です。



さて、今回作るライフゲームの内容を改めて整理しておきましょう。エージェントは格子空間を敷き詰めるセルです。各セルは、下図のフローチャートに従い、「生」と「死」の間で状態を変化させます。セルの初期状態は、さしあたりランダムに決めることにします。



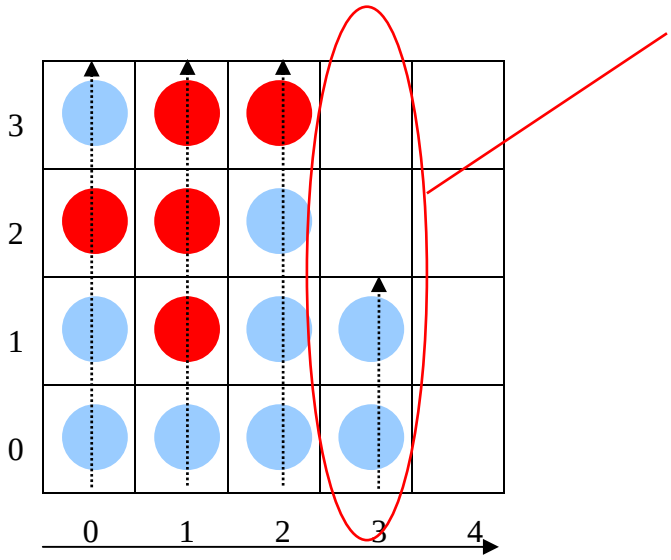
このようなモデルを作る上で学ばなければならない事項は大別して二点あります。

① For 文を使った格子空間でのエージェントの生成・配置

今回のエージェントは、格子空間をぎっしりと埋め尽くすセルです。このようなエージェントを配置する際に頻繁に用いられるのが、**For文のネスト**を利用したエージェントの生成・配置方法です。具体的な書式についてはモデルを作る過程で説明しますが、For文を二重に用いることによって、概略下図のようなイメージで、格子空間の各座標を順次巡っていき、各座標点でエージェントを一つずつ生成していくという方法です。

分居モデルや風邪引きモデルでは、特定数のエージェントをまず生成し、最後に RandomPutAgtSet 関数を用いてそれを一気に空間上にばらまくという方法をとっていました。それに対して、今回の配置方法は、空間上でエージェントを一つずつ作りながら置いていくというよりきめ細かなものになっています。この手法は、たとえば下図のように、エージェントの位置とその状態・属性（この場合は色）を関連づけて配置する際に力を発揮します。高度なエージェント配置方法のひとつとして、この機会に学んでおきましょう。

これに関連して、空間の幅および高さを取得する**GetWidthSpace 関数**および**GetHeightSpace関数**についても学ぶことにします。



② GetHistory 関数を用いた状態変化

前回作った風邪引きモデルでは、各ステップ、エージェントが空間を移動していき、移動先に風邪引きさんがいるなら、「その場で」風邪引きの状態になるというルールになっていました。これに対して、ライフゲームを含むセル・オートマトンでは、セルの現在の状態

が一期前の周囲の状態に依存して一斉に決まり、一期先の状態が現在の状態に応じて一斉に決まるというダイナミクスが展開していきます。ちょうど高校数学で学んだ漸化式ないし差分方程式のイメージ（x(t)=F(x(t-1))、x(t+1)=F(x(t))等）です。

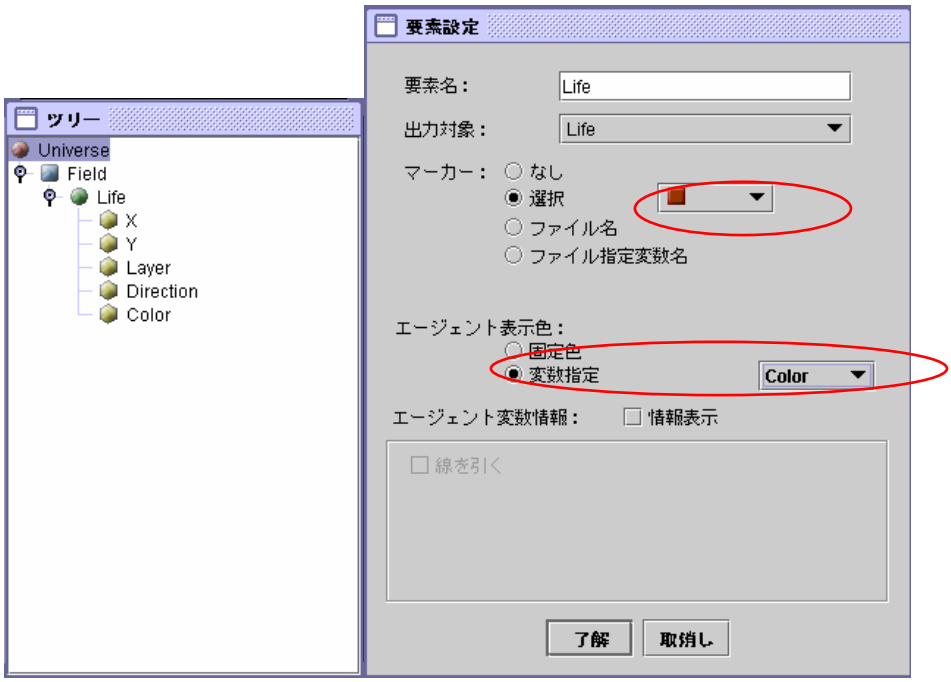
このような状態変化をKK-MASで表現する手法は幾つかありますが、ここでは、変数のプロパティで過去の変数値を参照できるようにした上で、**GetHistory関数**を用いて周囲のエージェントの一ステップ前の状態（「生」か「死」か）をチェックし、それに基づき自分の現在の状態（「生」か「死」か）を決めるという方法を学びます。これにより、全てのセルが一期前の周囲の状態に依存して「同時に」変化を繰り返していくセル・オートマトンのダイナミズムが生まれることになります。

ここで注意しなければならないのは、この「同時に」「一斉に」変化するというのが、あくまで概念上のもので、すでに前々回学んだように（資料６ページ参照）、計算機上は、そうはなっていないという点です。つまり、「１ステップ」という「一瞬」の間に、各エージェントが時間差を伴って逐次的にルールを実行しているわけです。逆に言えば、エージェントに同時に行動させる（これを「同期させる」と言います）モデルを KK-MAS で作りたい時に、この計算機上の時間差を無視して、たとえばエージェントの現ステップの状態を、実際には変化の途上にある当期の周囲の状態に依存して決めるようなルールを書いてしまうと、思わぬ相互作用をモデルに持ち込んでしまうことになります。GetHistory 関数を用いて、すでに変化が確定した一ステップ前の状態を参照する方法は、KK-MAS 上で「同期」を実現させるための基本的な手法の一つになっています。

●モデル作り１：エージェントの生成と配置

では、ライフゲームを作っていきましょう。まずは、いつものようにツリーでモデルの構造をデザインすることから始めます。Universe に 20×20 の空間（以下では Field）を追加し、後者にエージェント（Life）を加えましょう。前回のモデル同様、ライフゲームも、エージェントは「生」と「死」というふたつの状態を持ち、状態に応じて表示色を変化させることになりますが、状態と色とは一対一で対応しているので、今回は、色変数（Color）で状態の違いも表現することにします。変数型は整数型のままです。次ページ図左のようなツリーが出来上がったでしょうか。

次に、出力設定で二次元マップとエージェントの表示設定を行います。エージェント表示色は前回同様「変数指定」で、プルダウンから Color 変数を指定します（次ページ図右）。エージェントを表示させるマーカーもセルらしく四角形にしておきましょう。空間の右端と上端に出てくる「のりしろ」が気になる人は、「マップ出力設定」の「X 軸設定」「Y 軸設定」の「最大値」を 19 にするとよいでしょう。



さて、ここまで準備ができたら、初期状態を作るためのルールを記述します。

[1] 20×20 の二次元空間を Life エージェントで埋め尽くしていきます。Life の初期の状態はランダムに決め、三割が「生」の状態とし、これを赤色で表します。残りは「死」の状態で、これを水色で表します。

＞ポイント①で述べたことを念頭に置いて、Univ_Init に下記のように記述します。以下の例では、Agt_Init は用いていません（用いた書き方も可能です）。

```
Univ_Init{

        Dim i As Integer
        Dim j As Integer
        Dim cell As Agt

        // ↓各座標ごとに Life エージェントを生成し空間をぎっしりつめる
```

CreateAgt 関数で作ったエージェントを代入し参照するために用意しておく

```
        // X 方向の繰り返し
        For i = 0 To GetWidthSpace( Universe.Field ) - 1

                // 各 X 値について Y 方向の繰り返し
                For j = 0 To GetHeightSpace( Universe.Field ) - 1

                        // 生成した Life をエージェント型変数に格納
                        cell = CreateAgt( Universe.Field.Life )

                        // Life の座標を指定
                        cell.X = i
                        cell.Y = j

                        // 状態はランダムに決定
                        If Rnd( ) < 0.3 Then
                                cell.Color = Color_Red
                        Else
                                cell.Color = Color_Cyan
                        End If

                Next j

        Next i

}
```

空間の幅を取得

高さを取得

生成したエージェントをその場で代入

For 文の i, j を座標値として使用

まず、目につくのが、新しい関数 GetWidthSpace( )および GetHeightSpace( )です。書式は、

```
GetWidthSpace ( 空間名 )
```

```
GetHeightSpace ( 空間名 )
```

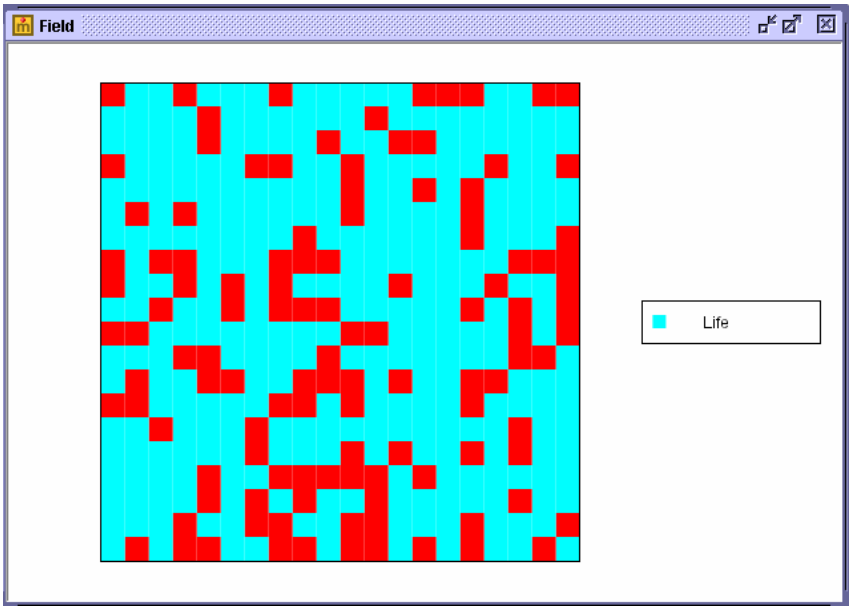
で、関数名から察しがつくように、それぞれ空間の幅（X 方向の長さ）および高さ（Y 方向の長さ）が返ってきます（整数型）。ルールの中で空間の幅や高さを用いる際、たとえば空間プロパティで空間の大きさを変える度に、ルールの中の数値をいちいち書き換えるのは

面倒です。両関数は、引数で指定した空間の大きさを自動的に判別してくれます。

次に、「**cell = CreateAgt(Universe.Field.Life)**」の部分です。これまで CreateAgt 関数は単独で用いてきましたが、この関数は返り値として生成したエージェント（個別エージェントなのでエージェント型変数）を返しています。エージェント型変数 cell に CreateAgt 関数で返ってきた新規エージェントを代入することで、このエージェントが持つ変数を参照したり設定したりすることが可能になります。その際、「cell.Color」といった書式を用いることは、前回説明した通りです。今回はこれを用いて、エージェントの初期の状態を Agt_Init ではなく Univ_Init で設定しています。

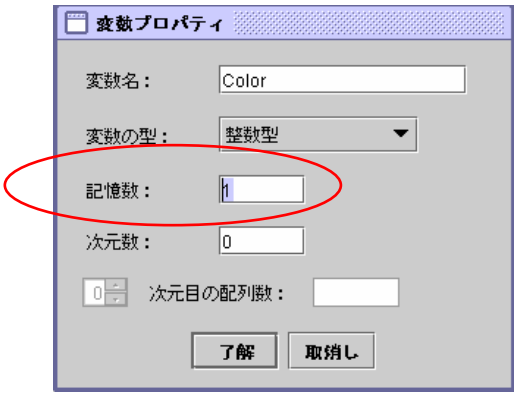
さて、上記ルールですが、全体が、For 文の中に For 文が入り込んだ入れ子構造（これを For 文の「ネスト」と言います）になっています。複雑に見えるかもしれませんが、４ページの図のイメージを思い浮かべれば、何を行っているか次第に分かってくるはずです。つまり、外側の For 文の各 i の値に対して、内側の For 文の j の値を 0 から変化させながら、CreateAgt 関数で繰り返しエージェントを生成しているわけですが、「cell.X = i」「cell.Y = j」とあるように、結局このルールは、「各 X の座標値に対して、Y 軸方向に繰り返しエージェントを生成・配置していき、それを全ての X の値に対して繰り返せ」という処理を実行していることになります。For 文の参照変数 i, j の上限値が GetWidthSpace 関数等で空間の大きさと関連づけられているのもこのためです。

ルールが完成したら、忘れずファイルを保存して、実行ボタンを押してみましょう。



●モデル作り２：エージェントのルールの記述

次にステップごとの Life エージェントの行動ルールを、冒頭のフローチャートに従って、Agt_Step に書き込んでいくわけですが、その前に一つだけ準備しておくことがあります。ポイント②で述べたように、ここでは GetHistory 関数を用いて一ステップ前のエージェントの状態変数（Color）の値を参照することになりますが、そのためにはこの変数の過去一期分の値を残しておくようあらかじめ設定しておく必要があります。具体的には、下図のように、Color 変数のプロパティ画面で、「記憶数」を「1」に変更してやります。



それでは Agt_Step のルールの記述に移りましょう。

[２] 各ステップ、Life は、視野１の範囲内にいる全ての Life エージェント（自分は除く）の１ステップ前の状態に基づき、現在の自分の状態を決めます。もし、「生」（つまり赤色）の Life の数が３つならば、自分の状態は「生」になり、２つならば、自分の状態は一期前から変化しません。それ以外の場合、自分の状態は「死」（水色）になります。

＞ルールは下記の通りです。

```
Agt_Step{

        Dim num_alive As Integer
        Dim neighbor As Agt
        Dim neighborhood As AgtSet

        // GetHistory 関数で０ステップ目が参照できないため一ステップ先送り
        If GetCountStep( ) > 1 Then
```

```
                // 周囲の Life エージェントを参照
                MakeOneAgtSetAroundOwnCell(  neighborhood,1,Universe.Field.Life,Fal
        se )

                // 前ステップで生きていた Life の数をカウント
                num_alive = 0
                For Each neighbor In neighborhood
                        If GetHistory( neighbor.Color, 1 ) == Color_Red Then
                                num_alive = num_alive + 1
                        End If
                Next neighbor

                // ↓一ステップ前の周囲の状態から現在の自分の状態を決定
                // 生きている Life の数がちょうど３なら「生」
                If num_alive == 3 Then

                        My.Color = Color_Red

                // 生きている Life の数が２なら一期前と同じ状態
                ElseIf num_alive == 2 Then

                        My.Color = GetHistory( My.Color, 1 )

                // それ以外は「死」
                Else

                        My.Color = Color_Cyan

                End If

        End If

}
```

GetHistory 関数を使って一期前の周囲の Life の状態を調べる

一期前の自分の状態をそのまま継承（何も書かなくてもよい）

まず、全体が「If GetCountStep( ) > 1 Then ～ End If」で囲まれ、ルールが２ステップ目から実行されるようになっていますが、これは GetHistory 関数の現段階の仕様によるものです。詳しい話は省略しますが、近日中に仕様が変更され、このような記述を行う必要がなくなりますので、ここではあまり深く考えずに If 文を書き加えておきましょう。

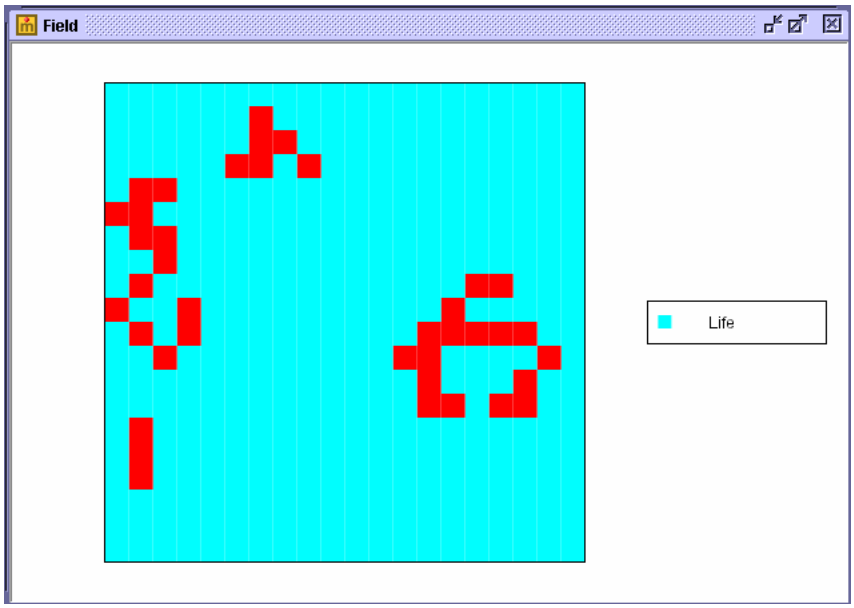
ルールの中身をさらに見ていくと、MakeOneAgtSetAroundOwnCell 関数で周囲のエージェントを集め、For Each 文で各々をチェックして、「生」の状態（赤色）のエージェントをカウントしています。構文上は、前回の風邪引きモデルにおける風邪引きエージェントのカウント方法とほとんど同じですが、ポイント②で述べたように、ここでは周囲のエージェントの１ステップ前の状態を参照する必要があります。そこで、GetHistory 関数の登場です。この関数は、

GetHistory(変数名（整数型・実数型など）, 整数型変数 n)

という書式で、第一引数の変数の n ステップ前の値を返します（変数プロパティで「記憶数」を n にしておく必要があります)。上記ルールでは、GetHistory(neighbor.Color, 1) によって、隣の Life エージェントの１ステップ前の Color 変数の値を取り出しているわけです。

後半では、If 文を用いて、１ステップ前の周囲の状態に応じて、自分の状態を決めるルールが書かれています。「My.Color = GetHistory(My.Color, 1)」の部分は、現在の自分の Color 変数に一期前の値を代入するという処理を表していますが、要するに状態が変化しないということなので、この部分には何も記述しなくても構いません。

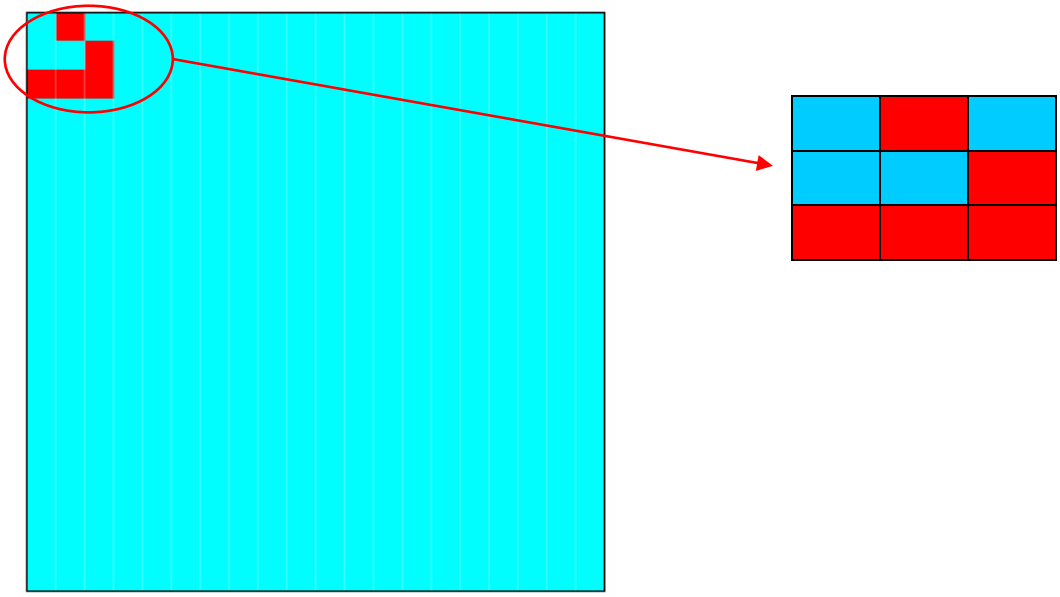
ここまででライフゲームの骨格は完成しました。ファイルを保存して繰り返し実行してみましょう。単純なルールから出てくる、きわめてバラエティに富んだ振る舞いにビックリしませんでしたか。



●モデル作り３：特定の初期配置の形成（時間があれば）

　エージェントの初期の配置のわずかな違いで全く異なるダイナミクスが出てくるのが、ライフゲームの面白さの一つです。これまで、興味深い状態変化のパターンを生み出す様々な初期状態の配置が「発見」され提起されてきました。以下で作る「グライダー」と呼ばれる配置もそのひとつです。どのような振る舞いが出てくるかは、作って実行してみてからのお楽しみです。

[３] ライフゲームを下図のような初期配置「グライダー」から始めてみましょう。



＞Univ_Init において、各エージェントの初期状態をランダムに決めていた部分を下記のように変更します。If 文が入り組んでいて一見錯綜としていますが、行っていることは、上図の初期配置に従い、赤色になる Life エージェントを座標値によって選別しているだけです。このようにエージェントの状態と位置とを、ランダムではなく、指定された形で関連づける場合、今回学んだ For 文のネストによる初期配置の方法を用いるのが便利です。

```
        // グライダー型の配置を作る
        cell.Color = Color_Cyan
        If cell.X == 0 Then
                If cell.Y == GetHeightSpace( Universe.Field ) - 3 Then
                        cell.Color = Color_Red
                End If
        ElseIf cell.X == 1 Then
                If cell.Y == GetHeightSpace( Universe.Field ) - 1 OR cell.Y ==
GetHeightSpace( Universe.Field ) - 3 Then
                        cell.Color = Color_Red
                End If
        ElseIf cell.X == 2 Then
                If cell.Y == GetHeightSpace( Universe.Field ) - 2 OR cell.Y ==
GetHeightSpace( Universe.Field ) - 3 Then
                        cell.Color = Color_Red
                End If
        End If
```

ルールが書き上がり、モデルが動けば、今日の課題はクリアです。お疲れさまでした。

●宿題

[４]世界にはライフゲームを愛するマニアが多数います。インターネットで彼らのサイトを閲覧し、各自興味を持った初期配置（「グライダー」以外にも「へび」「宇宙船」「R ペントミノ」などいっぱいあります）でモデルを実行してみましょう（ただし無限増殖型の振る舞いなど、空間的な制約からダイナミクスを再現できない初期配置もあります）。コントロールパネルで初期配置を切り替えるようにできればなおよいでしょう。