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 5 回　神様の登場と分居モデルの完成（11 月 4 日）

●概略

○前回宿題の解答例と解説
○Universe：ルールの実行順序
○For 文の説明
○CreateAgt の説明
○RandomPutAgtSetCell の説明
○MakeAgtSet の説明
○MergeAgtSet の説明
○コントロールパネルの設定
○宿題

●前回の宿題の解答例と解説

［4］［2］で作ったモデルを使用します。歩行者 A は歩行者 B の近接者が一人でもいれば右に 20 度曲がるように、歩行者 B は逆に歩行者 A の近接者が二人以上いれば左に 10 度、曲がるようにルールを書いてください。

＞下記のように記述します。

歩行者 A のエージェントルール（間違い易いところを赤字にしてあります）

```
Agt_Init{

// 初期値を設定します
My.X = 0
My.Y = rnd()*50
My.Direction = rnd() * 360
My.Speed = rnd()*0.5 + 0.5
My.Destination = 0
}

Agt_Step{
// 前に進みます
forward(My.Speed)

// 方向を修正します
If my.Direction > 0 and    My.Direction <180    then
            Turn(-3)
else
            Turn(3)
end if

// 周囲を認識します
MakeOneAgtSetAroundOwn( my.Neighbors, 1, Universe.二次元空間.WalkerB, False)
// 近接者を数えます
my.Neighbor_Num = CountAgtSet(my.Neighbors)
// 近接者がいたら方向を転換します
If my.Neighbor_Num >= 1 then
            turn(-20)
end if
}
```

Ｂの近接者が一人でもいれば、20 度右へ

歩行者 B のエージェントルール

```
Agt_Init{

// 初期値を設定します
My.X = 49
My.Y = rnd()*50
My.Direction = rnd() * 360
My.Speed = rnd()*0.5 + 0.5
My.Destination = 180
}

Agt_Step{
// 前に進みます
forward(My.Speed)

// 方向を修正します
If my.Direction > 0 and    My.Direction <180    then
            Turn(-3)
else
            Turn(3)
end if

// 周囲を認識します
MakeOneAgtSetAroundOwn( my.Neighbors, 1, Universe.二次元空間.WalkerA, False)
// 近接者を数えます
my.Neighbor_Num = CountAgtSet(my.Neighbors)
// 近接者がいたら方向を転換します
If my.Neighbor_Num >= 2 then
            turn(10)
end if
}
```

Ｂの近接者が二人以上いれば、10 度左へ

［5］50×50 のセル型空間上に、2 種類のエージェントをランダムに配置します（人数は任意)。歩行者 A は歩行者 B の近接者が一人でもいれば止まり、それ以外は空きスペースに一歩ずつ移動します。歩行者 B は逆に歩行者 A の近接者が二人以上いれば止まり、いなければ同様に空きスペースに移動するルールを書いてください。

＞下記のように記述します。

歩行者 A のエージェントルール

```
Agt_Init{

// 初期値を設定します
My.X = int(rnd()*50)
My.Y = int(rnd()*50)



}
Agt_Step{

// 周囲を認識します
MakeOneAgtSetAroundOwn( my.Neighbors, 1, Universe.二次元空間.WalkerB, False)
// 近接者を数えます
my.Neighbor_Num = CountAgtSet(my.Neighbors)

// 近接者がいたら止まります
If my.Neighbor_Num >= 1 then
        forwardxcell(0)
        forwardycell(0)

//それ以外は空きスペースに移動します
else
        movetospaceowncell(1)

End if
}
```

ツリーで、エージェント変数として定義しておきます。

歩行者Ｂの数が 1 以上いれば、どこにも動きません。つまり forwardcell の引数は 0 になります。

歩行者 B のエージェントルール

```
Agt_Init{

// 初期値を設定します
My.X = int(rnd()*50)
My.Y = int(rnd()*50)

}
Agt_Step{
// 周囲を認識します
MakeOneAgtSetAroundOwn( my.Neighbors, 2, Universe.二次元空間.WalkerA, False)

// 近接者を数えます
my.Neighbor_Num = CountAgtSet(my.Neighbors)
// 近接者がいたら止まります
If my.Neighbor_Num >= 2 then
        forwardxcell(0)
        forwardycell(0)

//それ以外は空きスペースに移動します
else
        movetospaceowncell(1)

End if
}
```

［6］10×10 のセル型空間上に、2 種類のエージェントをランダムに配置します（人数は任意）。どちらの種類も、自分と同じタイプのエージェントの近接者が二人以上いれば空きスペースに移動し、いなければ止まるルールを書いてください。分居モデルらしくなりましたか？

＞下記のように記述します（解説は省略します）。

WalkerA のエージェントルール

```
Agt_Init{

// 初期値を設定します
My.X = int(rnd()*10)
My.Y = int(rnd()*10)

}
Agt_Step{

// 周囲を認識します
MakeOneAgtSetAroundOwn( my.Neighbors, 1, Universe.二次元空間.WalkerA, False)
// 近接者を数えます
my.Neighbor_Num = CountAgtSet(my.Neighbors)

// 近接者がいたら空きスペースに移動します
If my.Neighbor_Num >= 2 then
movetospaceowncell(1)

//それ以外は止まります
else

        forwardxcell(0)
        forwardycell(0)
End if

}
```

WalkerB のエージェントルール

```
Agt_Init{

// 初期値を設定します
My.X = int(rnd()*10)
My.Y = int(rnd()*10)

}
Agt_Step{


// 周囲を認識します
MakeOneAgtSetAroundOwn( my.Neighbors,1, Universe.二次元空間.WalkerB, False)

// 近接者を数えます
my.Neighbor_Num = CountAgtSet(my.Neighbors)

/// 近接者がいたら空きスペースに移動します
If my.Neighbor_Num >= 2 then
movetospaceowncell(1)

//それ以外は止まります
else

        forwardxcell(0)
        forwardycell(0)
End if

}
```

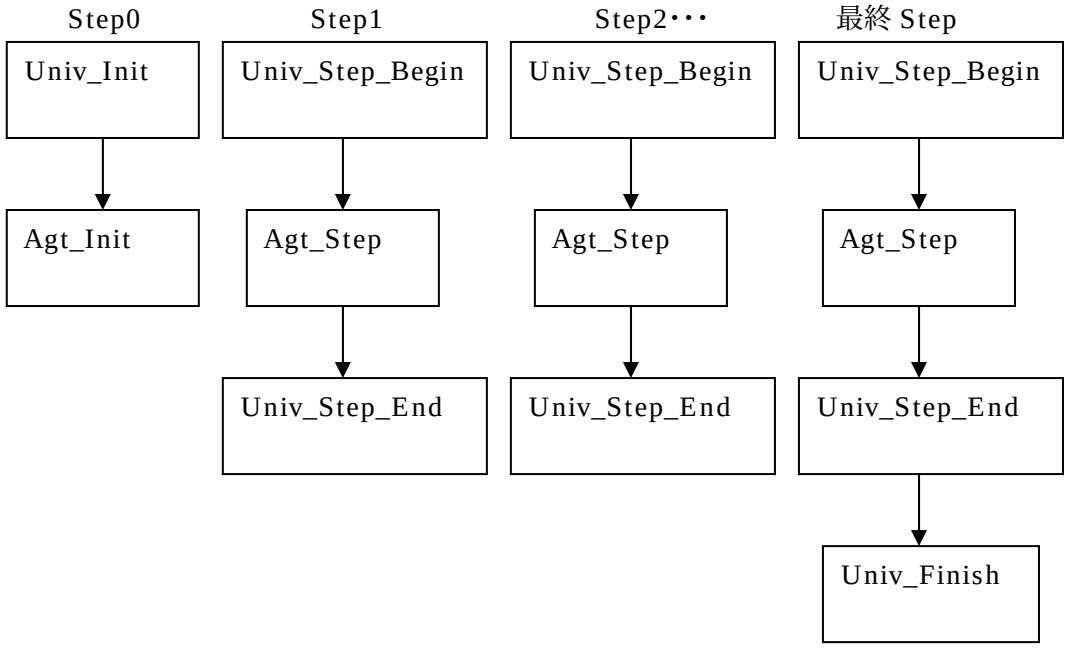
今回はT・シェリングの分居モデルを作成してもらいます。

●Universe：ルールの実行順序

Universe のルールエディタにルールを書き込むことで、いろいろなことを実行させることができます。その実行順序について説明しておきます。エディタを開けてみてください。Universe のルールエディタは、Univ_Init{}、Univ_Step_Begin{}、Univ_Step_End{}、Univ_Finish{}の４つの部分に分かれています。

Univ_Init{}	シミュレーションの最初に一度だけ実行される
Univ_Step_Begin{}	各ステップの初めに実行される
Univ_Step_End{}	各ステップの終わりに実行される
Univ_Finish{}	シミュレーションが終了するときに一度だけ実行される

つまり実行順序は、以下のようになります。



●今日の文法

☆**For 文**（繰り返し文）

If 文（条件文）と並ぶ基本命令文です。指定した回数だけ、命令を繰り返させるのに用います。以下のように書けば、ルール A を 10 回繰り返します。

```
dim i as integer
For i = 0 to 9
        ＜ルールA＞
Next i
```

☆**CreateAgt**　　エージェントを創り出す

指定した種類のエージェントを一人創り出します。１つの条件（＝引数）を設定してやります。

```
CreateAgt( 創り出すエージェント種別 )
```

Ex. CreateAgt(Universe.Streets.RedTurtle)

☆**RandomPutAgtSetCell**　　集合に含まれるエージェントをランダムに配置する

指定したエージェント集合に含まれているエージェントを空間内にランダムに配置する関数です。セル型に配置するために、Cell はセル型に配置することを意味します。

```
RandomPutAgtSetCell( エージェント集合型変数, 重なりを許すか（True/False） )
```

Ex. RandomPutAgtSetCell(Universe.Turtles, False)　　＞ツリーで設定するとき

Ex. dim Turtles as AgtSet

RandomPutAgtSetCell(Turtles, False)　　＞ルールで設定するとき

（まえもってルールの中で（先頭で）宣言しておく必要があります）

☆**MakeAgtSet**

指定された種類のエージェント全部のリストをエージェント集合型変数に格納する

先週は、MakeOneAgtSetAroundOwnというエージェント集合型関数を学びました。自分自身の周りにあるエージェントのリストをエージェント集合型変数の中に格納する変数でした。このようにさまざまな条件をつけずに、ある種類のエージェントを全部、変数の中に格納してしまう関数が、MakeAgtSetです。

```
MakeAgtSet( 生成されるエージェント集合型変数, エージェント種別)
```

Ex. MakeAgtSet(Universe.Turtles, Universe.Streets.RedTurtle)　　＞ツリー設定

Ex. dim Turtles As AgtSet

MakeAgtSet(Turtles, Universe.Streets.RedTurtle)　　＞ルール設定

☆**MergeAgtSet**　　エージェント集合を合体させる

あるエージェント集合型変数に、他のエージェント集合型変数に含まれているエージェントのリストを加えてしまう関数です。２つの条件を設定してやります。変数２に含まれているエージェントのリストを変数１に格納することができます。

MergeAgtSet(エージェント集合型変数１, エージェント集合型変数２)

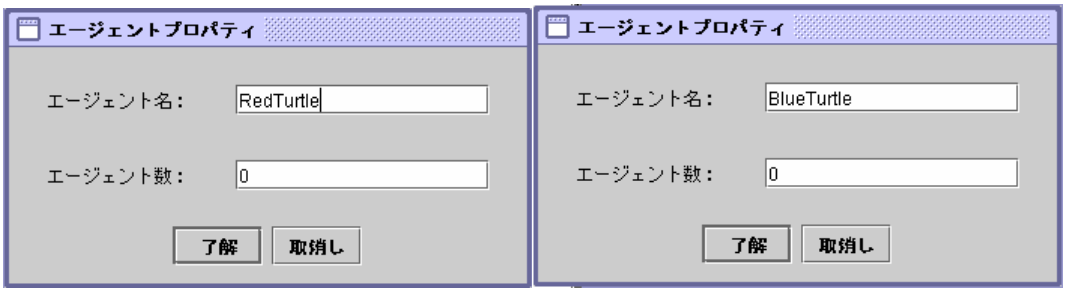
Ex. MergeAgtSet(Universe.Turtles1, Universe.Turtles2)　　＞ツリー設定

Ex. dim Turtles1 As AgtSet
dim Turtles2 As AgtSet
MergeAgtSet(Turtles1, Turtles2)　　＞ルール設定

例題で確認してみましょう

[１] 20×20 のセル型空間上に、2 種類のエージェント（RedTurtle と BlueTurtle）をそれぞれ 10 ずつ生成し、ランダムに、重ならないように配置します。

エージェントを追加する際、エージェントの数は０にしておきます。



出力設定でマップ出力をした後、Universe のルールエディタを開きます。

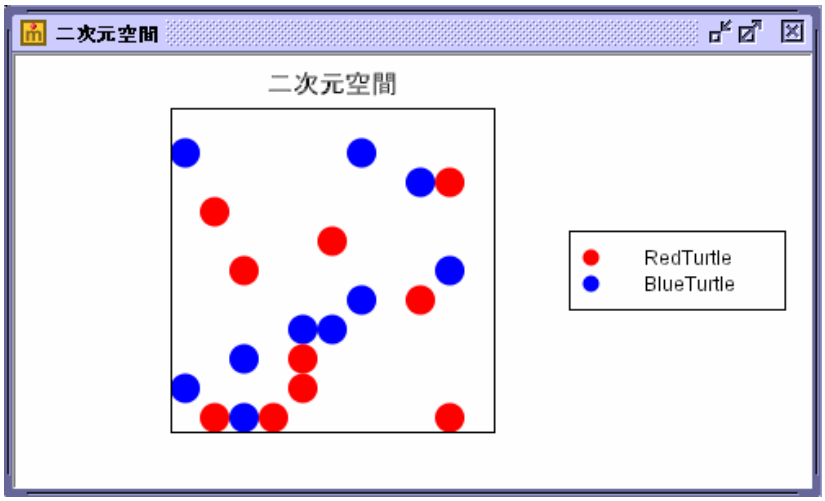
＞下記のように記述します。

もちろん、この記述のやり方が唯一というわけではありません。色々と（他の関数も使用しつつ）試してみましょう。

```
Univ_Init{
Dim i as integer
Dim RTurtles As AgtSet
Dim BTurtles As AgtSet
Dim AllTurtles As Agtset

// 2 つのエージェントを 10 ずつ発生させる
For i = 0 to 9
        CreateAgt(Universe.二次元空間.RedTurtle)
        CreateAgt(Universe.二次元空間.BlueTurtle)
Next i

// RTurtles という変数に赤亀のリストを格納する
MakeAgtSet(RTurtles, Universe.二次元空間.RedTurtle)
// BTurtles という変数に青亀のリストを格納する
MakeAgtSet(BTurtles, Universe.二次元空間.BlueTurtle)


//AllTurtles という変数に RTurtles という変数（赤亀のリスト）の値を加える
MergeAgtSet(AllTurtles, RTurtles)
//AllTurtles という変数に BTurtles という変数（青亀のリスト）の値を加える
MergeAgtSet(AllTurtles, BTurtles)

//AllTurtles に含まれているエージェント（全ての亀）をランダムに配置する
RandomPutAgtSetCell(AllTurtles, False)
}
Univ_Step_End{
}
Univ_Finish{
}
```

0 もカウントされるので、10 人作るときは 0 to 9　と記述する。

ちゃんとランダムに、重ならずに配置されましたか？



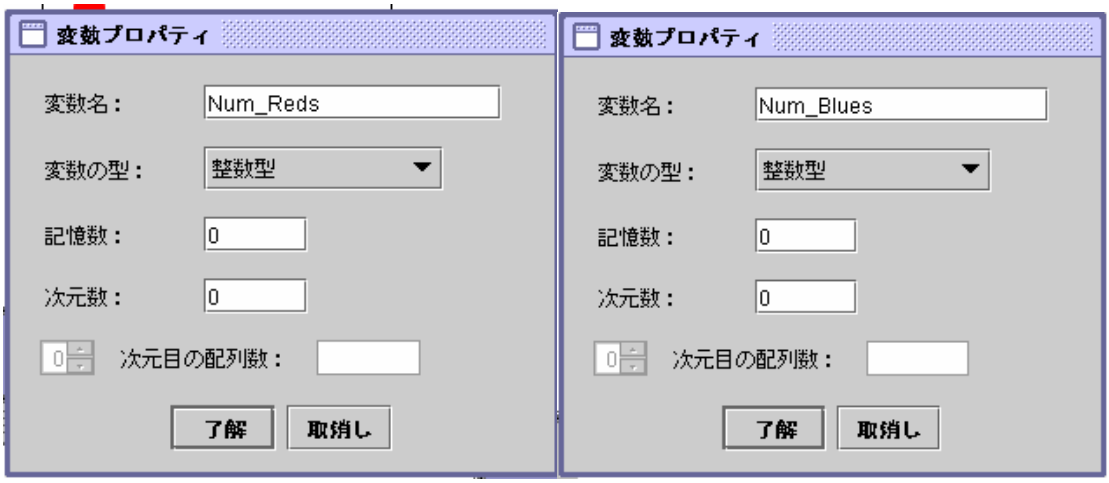
●コントロールパネルを作る

さきほどのルールでは、”i = 0 to 9”とわざわざルールエディタに記述しましたが、モデルにコントロールパネルを設定することで、動かしているモデルの設定を外からいじることができるようにすることが出来ます。コントロールパネルに設定することができるのは、Universe レベルの変数のみです。

[２]RedTurtle と BlueTurtle の数を表す変数を Universe に設定し、それをコントロールパネルで設定できるようにしてみましょう。

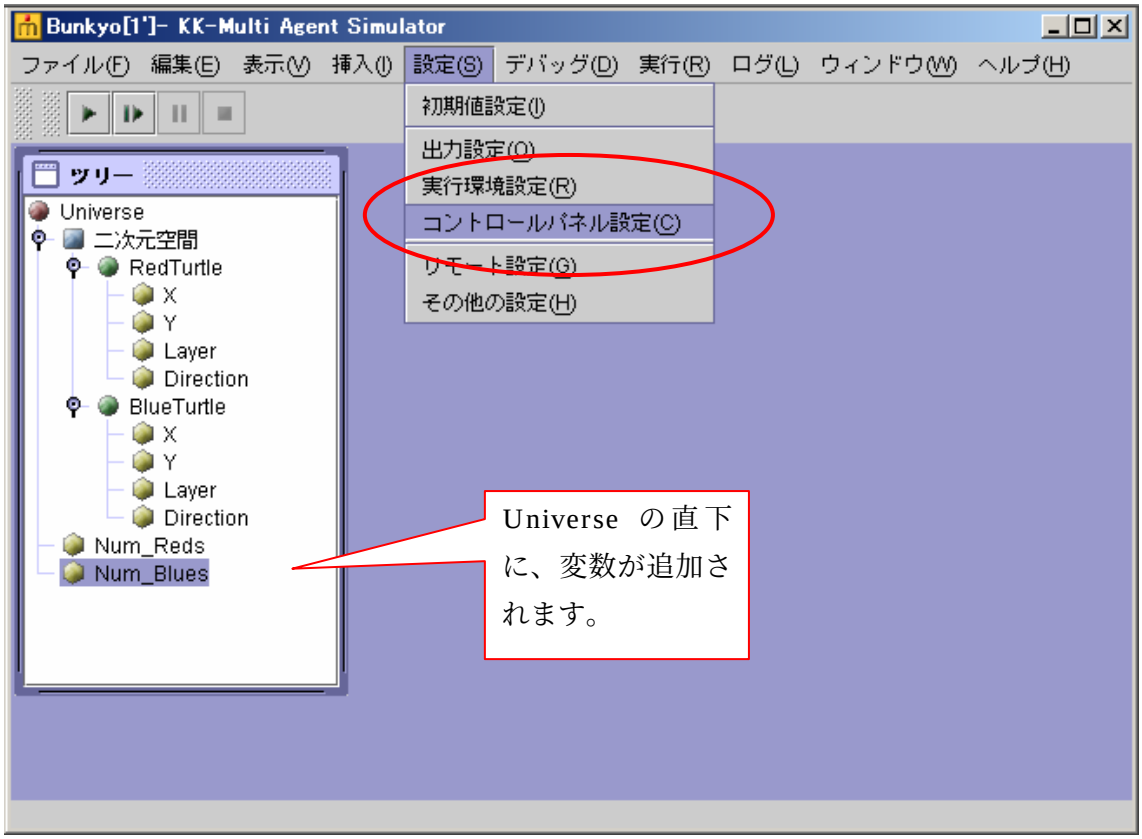
＞まず初めに、Universe を右クリックし、変数の追加を選択します。

変数は 2 つ（Num_Reds と Num_Blues）追加します。

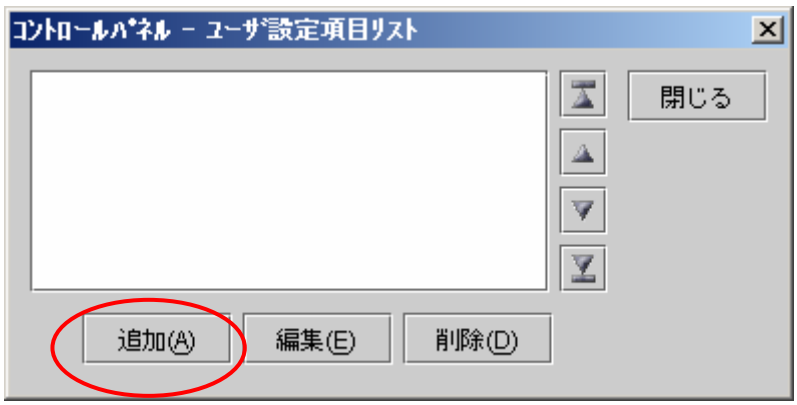


＞次はコントロールパネルです。

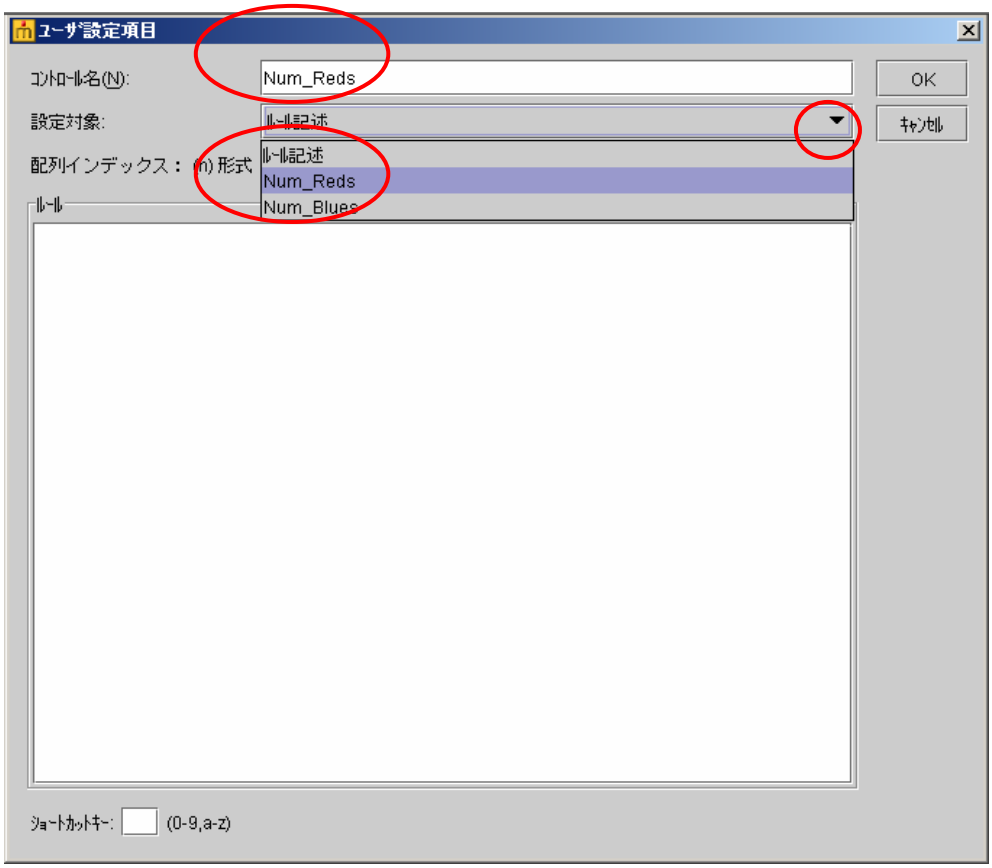
［設定＞コントロールパネル設定＞追加］で設定します。



追加をクリックします。

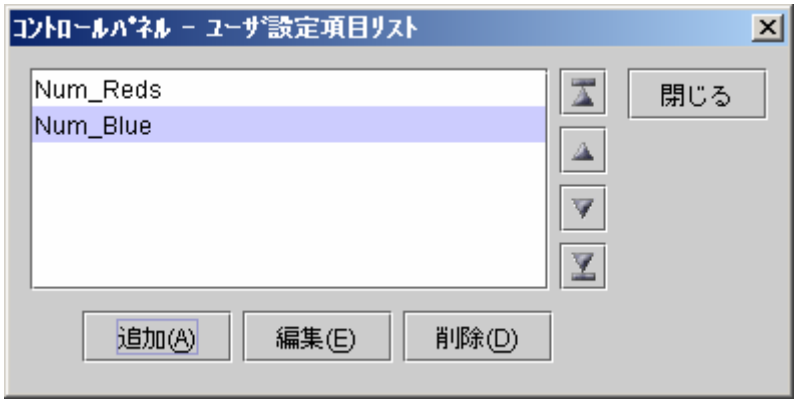


コントロール名に任意の名前（今回は変数と同じ名前）を記入して、Universe の下につくった変数を選択します。

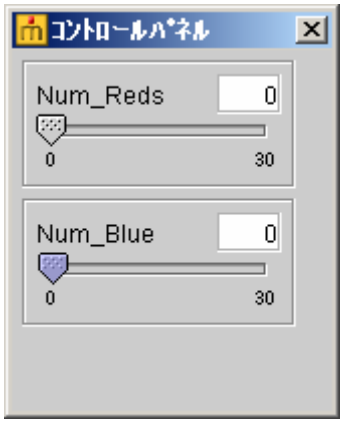


するとインターフェースが選択できるようになります。エージェントの数を増減させやすいように、スライドバーにチェックをつけて、範囲を指定します。目盛り間隔も自由に指定できます。

Num_Blue も同様に設定してあげてください。そうするとコントロールパネルユーザ設定項目リストに、２つ追加されます。



閉じるをクリックしてから、［表示＞コントロールパネル］を選択すると、コントロールパネルが出現します（ただしまだルールには反映されません）。



コントロールパネルでの設定を、ルールに反映したければ、ルールに記述する必要があります。Universe のルールエディタを開いて、エージェントを発生させるルールを下記のように変更してください。

```
/*
// 2 つのエージェントを 10 ずつ発生させる
For i = 0 to 9
        CreateAgt(Universe.二次元空間.RedTurtle)

        CreateAgt(Universe.二次元空間.BlueTurtle)
Next i
*/

// 赤亀を発生させる
For i = 0 to universe.Num_Reds - 1
        CreateAgt(Universe.二次元空間.RedTurtle)
Next i
//青亀を発生させる
For i = 0 to universe.Num_Blues -1
        CreateAgt(Universe.二次元空間.BlueTurtle)
Next i
```

［1］のルールはブロック単位でコメントアウトしてしまいましょう。
/*　～　*/　でコメントアウトできます。

0 ステップ分カウントされますので、コントロールパネルの数を出すときは必ず 1 を引く必要があります。

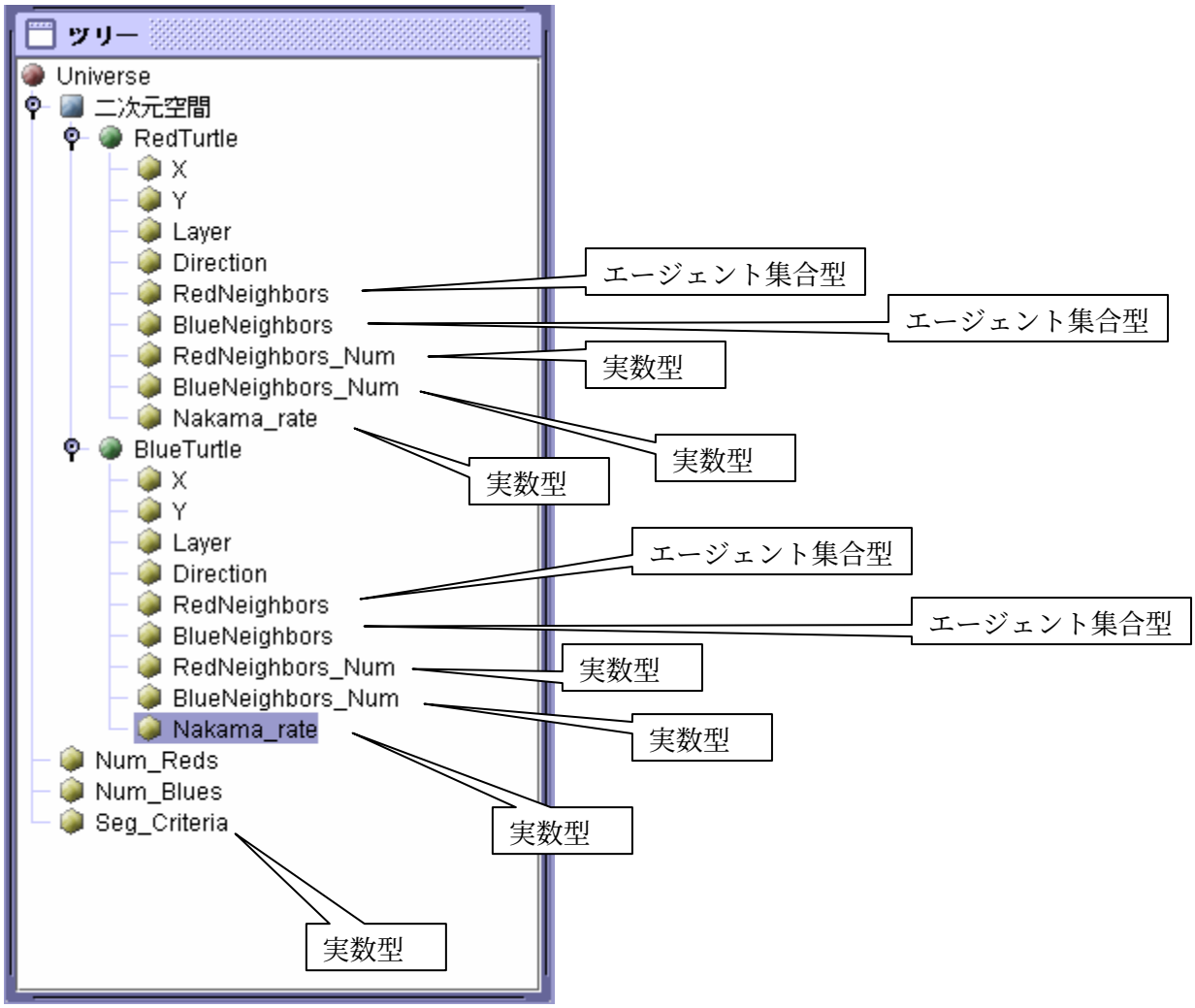
コントロールパネルで数を適当に設定して実行してください。

＞次に、エージェントのルールに移ります。

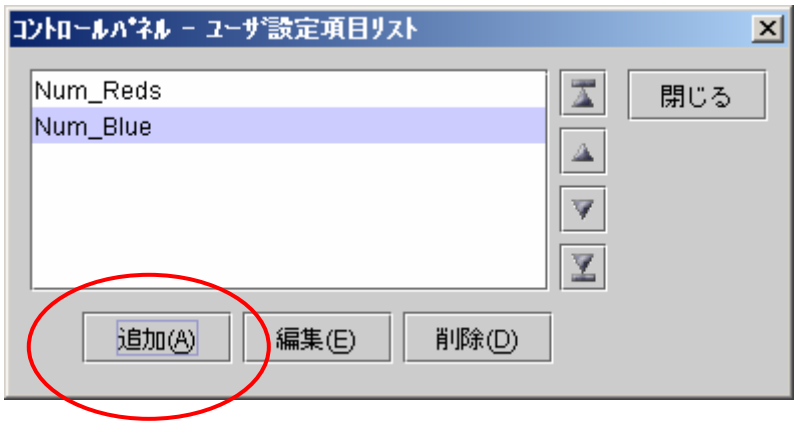
[３]不満があれば移動するルールを亀に書いてみます。

それぞれの亀に自分の分居率を計算させます。そして一般に亀が満足する分居率の基準（満足水準、閾値）を設定してやり、それと比較して不満か、どうかを決めます。この満足水準を Universe の変数として設定してやると、コントロールパネルで操作することが出来て簡単です。

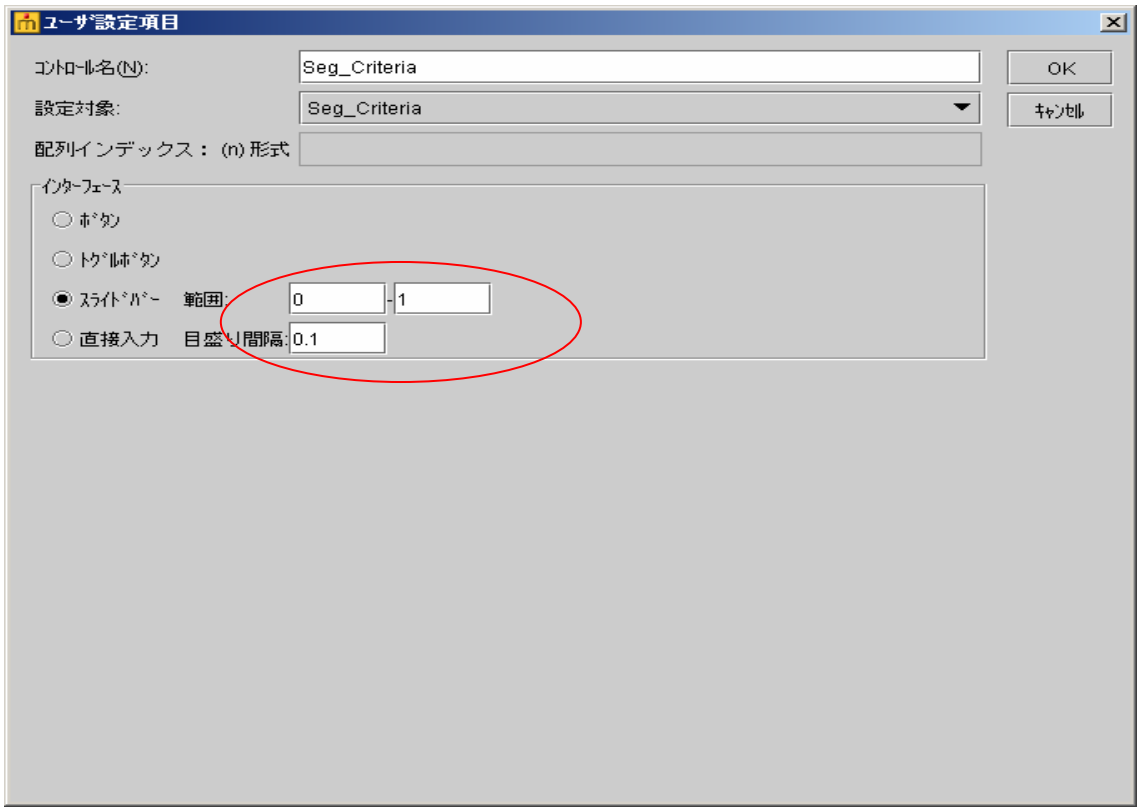
まず、各エージェントにエージェント集合型の変数 2 つと実数型の変数 3 つを追加します（名前は任意。ここではそれぞれ、RedNeighbors、BlueNeighbors、RedNeighbors_Num、BlueNeighbors_Num、 Nakama_rate とします)。また、Universe の下に実数型の変数を追加します（こちらも名前は任意。ここでは Seg_Criteria としています)。



コントロールパネルで Seg_Criteria を設定できるようにしましょう。



インターフェースはスライドバーにして、範囲は 0～１、目盛り間隔は 0.1 に設定してください。



エージェントルールを、下記のように記述します。

RedTurtle のルール

```
Agt_Init{
}

Agt_Step{
dim AllNum as double

//赤亀を数える
MakeOneAgtSetAroundOwnCell(My.RedNeighbors, 1, Universe.二次元空間.RedTurtle, False)
My.RedNeighbors_Num = CountAgtSet(My.RedNeighbors)

//青亀を数える
MakeOneAgtSetAroundOwnCell(My.BlueNeighbors, 1, Universe.二次元空間.BlueTurtle, False)
My.BlueNeighbors_Num = CountAgtSet(My.BlueNeighbors)

//周りのすべての亀の数を計算します
AllNum = My.RedNeighbors_Num+My.BlueNeighbors_Num

//仲間率を計算します
If AllNum = = 0 then
        My.Nakama_Rate = 0
else
        My.Nakama_Rate = My.RedNeighbors_Num / AllNum
end if

//不満なら移動します
If My.Nakama_Rate < Universe.Seg_Criteria then
        MoveToSpaceOwnCell(2)
end if

}
```

後で亀の合計を計算するために実数型の変数を定義しておきます。

このルールは下記「割り算の掟」参照

BlueTurtle のルール

```
Agt_Init{
}

Agt_Step{

dim AllNum as double

//赤亀を数える
MakeOneAgtSetAroundOwnCell(My.RedNeighbors, 1, Universe.二次元空間.RedTurtle, False)
My.RedNeighbors_Num = CountAgtSet(My.RedNeighbors)

//青亀を数える
MakeOneAgtSetAroundOwnCell(My.BlueNeighbors, 1, Universe.二次元空間.BlueTurtle, False)
My.BlueNeighbors_Num = CountAgtSet(My.BlueNeighbors)

//周りのすべての亀の数を計算します
AllNum = My.RedNeighbors_Num+My.BlueNeighbors_Num

//仲間率を計算します
If AllNum == 0 then
        My.Nakama_Rate = 0
else
        My.Nakama_Rate = My.BlueNeighbors_Num / AllNum
end if

//不満なら移動します
If My.Nakama_Rate < Universe.Seg_Criteria then
        MoveToSpaceOwnCell(2)
end if

}
```

〇割り算の掟

一、ゼロで割ってはいけません。

周りに誰もいない亀は、分母が０になってしまいます。あらかじめ、周りいる亀が０の亀は、自動的に分居率が０になるように、If 文で設定してあげましょう。

一、整数［割る］整数は自動的に、整数になってしまいます。

コンピュータプログラムの世界では、整数型変数 / 整数型変数は、自動的に整数として計算することになっています。なので、そのままだと分居率はいつも０になります。亀の数（赤亀の数、青亀の数）を実数型に設定してあげましょう。

★これで分居モデルは完成です！ちゃんと動くことを確認しましょう。

それでは宿題にまいります。今回は作成した分居モデルを改造しましょう。

［４］赤亀と青亀の持つ満足水準（Seg_Criteria）をコントロールパネルで別々に設定できるようにしてみましょう。

［５］分居モデルは、一定の比率の同種別の亀がいないと引っ越しするというルールでした。では、ある一定の異種別の亀がいると、嫌がって引っ越しするというルールを作ってみましょう。