Workshop on World Modeling・Workshop on Methods of Human Security Studies
2005 Summer Semester

Presiding Professor: Susumu Yamakage
TA; Takuto Sakamoto, Kazutoshi Suzuki,
Hiroyuki Hoshiro, Katsuma Mitsutsuji
Kazuya Yamamoto

Lecture Twelve: Streamline Complex Agents (July 12th)

Today's Target:
If you make agents do complex processing, the rules get complex and that makes it hard to read. Thus, this time we will discuss how to consolidate and streamline the rules. We will also discuss how to intricately designate execution order and how to synchronize agents.

● Write down complex rules in a simple fashion.
If you come to make models of a certain size, there are many instances where you would have to repeat the same processing over and over again. Especially when the overlapping parts contain complex nesting of conditional statements, it would be hard to read. And it may lead to detrimental errors and mistaking the number of if and endif. In such cases, by using subroutines and functions you can consolidate the duplicated parts at one place and be able to call them to order quite easily.

＊ Functions (user defined functions)
In cases where you need a return value (value you want to use again that was used inside a calculation and has returned to its original rule) there is a function. Stipulate the following after closing Agt_step by「 } 」at the very end of the rule:

Function< name of function>(parameter declaration)As <type of function return value>
{
< variable declarations section >
<execution section>
Return (name of variable) ← You can have as many as you like. You can return a

specified value per differentiation.

}

For example, if you want to create AgtSet in the foreground;

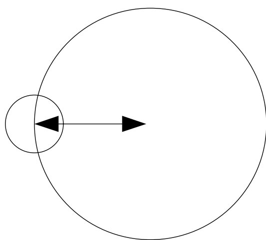
```
Function countagtsahead(distance as Double) As Integer
{
dim set as agtset
dim kazu as integer

forward(distance)                              ← proceed the designated distance
MakeAllAgtsetAroundOwn(set, distance, False)   ← create set
kazu = Countagtset(set)                        ← count number of agents
forward(-distance)                             ← return to original position
Return(kazu)                                   ← return what's inside kazu
}
```

Then you can list the agents inside the figure below, the round range. According to the rules of Agt_Init and Agt_Step,this is how it will appear.

```
if countagtsahead(1.2) > 0 then       ←if from1.2 frontward within a radius of 1.2
                                        there is something.
        my.speed = 0                  ← speed down to zero
    end if
```

It can now be used in the same manner as other functions. As long as there are no mutual reference user-defined functions can be used with the definition of other functions.



＊ Subroutine (consolidate a part of the rule)

If you want to simply process the overlapping, use this subroutine:

```
Sub<name>(parameter declaration) {
<variable declarations section>
<execution section>
}
```

Then it will be called with「name ()」inside the rule. Write in () of the parameter declaration only when there is a value that needs to be handed down. Let's say you made the following subroutine:「Proceed when there is empty space in front. Decide on your own the distance you are to proceed and at which range of view you will judge as being empty space.」

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

dim set as agtset

forward(dist)                              ←first of all proceed 1
makeallagtsetaroundown(set, sight, false)← look around after having proceeded
if countagtset(set) > 0 then               ← what if there is something
        forward(-dist)                     ← retrieve
end if
}
```

If you next input forwardspace(1,1) with Agt_Step, the following order will be executed;「Proceed forward by 1, if there is no one within 1 space of that space stay. If there is someone within one space of that space, return to the original position.」There is no return value in a subroutine. This means that unlike functions, we can not find a replacement with,「if processing 1() > 0 then ···」So in the following case, we must use a function;「We are at the edge of space and are unable to move, in this case, we are ・ 1」

Example ①

Let's make a space of 50×50 (no loop). In this space things fall right. With each step

from the bottom left a single ball appears and falls towards the right side. If there are no obstacles, the ball will keep on falling till the edge, one every step. If there is a ball to the right of this ball, if there is open space above, it will slide there. Let's write this function. We will create one agent every time in the Universe.

```
Agt_Step{
if CountAgtAhead(0.5) == 0 then        ← if there is nothing within a radius of 0.5 from
                                         0.5 ahead･･･
        forward(1)                     ← will proceed 1
else                                   ← if there is something
        turn(90)                       ←will look upward
        if CountAgtAhead(0.5) == 0 then   ←if nothing here,
        forward(1)                     ← will move upward
        end if
        turn(-90)                      ← will return to original direction
end if
}
```

Bonus: If you are chasing something and you've used Forward() or when you are heading somewhere, then there arises a need to know the direction of your destination. You will use arctangent but this could be a bit tricky With a lot of conditional statements. I have uploaded a function that will help you in such situations, please copy it.

● Execution Order

＊ The order in which the agent executes so and so; Execution Order

Though it may seem to be not so important, the execution, order, in fact, proves to be one of the key settings. Though on the surface it may seem as if everything is going well, there are instances where, in fact things exactly the opposite to what you have imagined are taking place. Execution order can be established in detail with 「Set＞Execution Environment Setting＞Execution Order」.At present the following four types of execution orders can be designated:

･1. Random

All agents are shuffled regardless of the type of agent.

・2. Random (change execution order on the first time only)

Only at the first time, regardless of the type, all agents are shuffled. From the second step, the execution will be identical to the first.

・3. Random (designation of the type of agent)

Execution order can be designated depending on the type of agent. The agents of the same type are shuffled at every step. The numbers on the left side show the execution order.

1, wolf
2, sheep
2, goat

Then the wolf comes first, and later the sheep and goats are executed all mixed together.

・4. Random by agent type (change execution order on the first time only)

Here you can designate an execution order for each type of agent. You can also shuffle the execution order within the agents of the same type for the first time only.

This option of being able to shuffle on the first time only is relatively important. Let's say that ■ is a running away model and that ▲ is a chasing model. Then the order will be such;

Even if the speed of ▲■ is the same, the distance between them does shrink and widen.

In this way the first order must be obeyed.

＊　Techniques to adjust the execution order.

・ Flag

When it comes to complex models that respond to the reactions of the opponent, there are cases where one line of action can not be written in one step alone.　Let's say three steps are necessary to complete a line of action. One turn after every three steps and it would be necessary to repeat this. In this case a Flag is used. We will make a variable that represents a phase, right under the Universe. We will write down a rule, 0→1→2→0→1→2→0, so that it will change.

```
universe.phase = universe.phase + 1
if universe.phase == 3 then
        universe.phase = 0
end if
(Or,universe.phase = getcountstep() mod 3)
```

And, in the agent rule, we will write

```
if universe.phase == 0 then
        rule of phase 1
elseif universe.phase == 1 then
        rule of phase 2
else
        rule of phase 3
end if
```

By making subroutine of the rules of each phase, it does become simpler and more comprehensible.

・ Skipping　steps

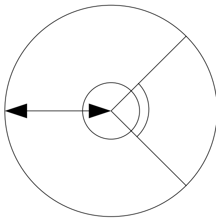
If you attempt to refer to memory of past ten turns, AND if you are still short of eleven steps, you will be referring to the memory before the 0 step. This would be a problem because you would be referring to a memory of a world before it came into being. This can be also dealt with the "flag", by enabling the rule only after eleven steps.　As to that which is short of eleven steps, you can simply skip the rule or execute an entirely different rule instead. With the same method, we can call for initial positioning or modify conditions in the course of a run.

Assignment

① Using the bonus function, please make a model where two agents chase each other around. The space is better looped. The chasing agent chases automatically. The escaping agent escapes in a straight line from a randomly-chosen place toward a randomly-chosen direction. The speed is 1 for both. (Set the garbage collection at 1 in Execution Environment Setting. If the speed is too fast, put a value after “wait for” and adjust with effective wait. You will use Getagt or For each. Be very careful about your execution order. The agent escaping goes first.)

②Make it possible to operate the direction of the escaping agent with the control panel. Also make them start from the same place. The agent chasing will start after waiting 10 steps.

③Study the figure below and make a function that looks forward at one angle of vision.



angle

dist