ホムンクルス人工知能スクリプトマニュアル　ver.1.0

目次

1 - 紹介
2 - スクリプト作動構造
3 - ラグナロクオンラインクライアント内部関数解説
4 - ラグナロクオンラインクライアント内部定数値解説
5 - デフォルトで提供するスクリプトの説明

1 - 紹介

ラグナロクオンライン（以下、「RO」） ゲーム内でのホムンクルスの行動は、RO クライアントプログラムが設置されているフォルダの AI フォルダの中にある AI.lua､Util.lua ファイルによって制御されています。この AI.lua ファイルはホムンクルスの人工知能（以下、「AI」）を制御するスクリプトが記述してあり、このスクリプトを変更することで、ホムンクルスの行動を自分が望むようにカスタマイズすることができます。AI.lua ファイル内のスクリプトは、RO クライアントと連動する一種のプログラムです。文法的な間違いや論理的な間違いがあった場合は、作動しません。なお、スクリプトにはルア（LUA）という言語を使用しています。

2 - スクリプト作動構造

RO クライアントは、クライアントプログラムでホムンクルスが生成された際、AI.lua、Util.lua ファイルを認識して AI を作動させます。

ホムンクルスが生成されるのは、次の場合です。

1) ホムンクルスを[コールホムンクルス]で誕生させた時
2) 戦闘不能になったホムンクルスを[リザレクションホムンクルス]で復活させた時
3) ホムンクルスを所有したキャラクターを選択してゲームを始めた時
4) ホムンクルスを所有したキャラクターが、ハエの羽、蝶の羽を使った時
5) ホムンクルスを所有したキャラクターが、ワープポイントから移動した時
6) ホムンクルスを所有したキャラクターが、カプラ職員の空間移動サービスを使った時

上の事項の共通点は、「ホムンクルスがマップに新たに現われた時」です。

AI.lua ファイルが認識された後に、RO クライアントプログラムは AI.lua スクリプトで AI（id）関数を実行します。id はゲーム上でのホムンクルスの固有番号です。id は RO クライアントが AI スクリプトで伝達する数字です。AI(id) の内容を編集して AI を変化させることができます。
基礎的なホムンクルスの行動、すなわち移動、攻撃、エサを食べる、スキル使用等の関数は、RO クライアントプログラムに内蔵しています。これらの関数を使用して、ホムンクルスの AI をカスタマイズしてください。

Const.lua　RO クライアントプログラム内の各種定数値に対する情報を取り扱っています。
AI.lua Util.lua で参照します。

Util.lua　リスト資料構造、いくつかの単純な機能の計算関数などが存在します。
AI.lua で参照します。

ホムンクルス AI の必須条件は AI.lua ファイルと AI.lua ファイル内に定義された AI（id）関数です。この 2 つの条件以外は全て選択事項ですので、厳密には Const.lua、Util.lua の 2 つは必要ありません。しかしスクリプトを作成する際にはさまざまな数値を扱いますので、活用すると便利です。

3 - ラグナロクオンラインクライアント内部関数解説

id : ゲーム内の物体が持つ固有番号

1) MoveToOwner (id)
　id　　: ホムンクルスの id
　戻り値 : なし
　機能　 : ホムンクルスを召喚者の近くに移動させる

2) Move (id, x, y)
　id　　: ホムンクルスの id
　x　　: 目的地 X 座標
　y　　: 目的地 Y 座標
　戻り値 : なし
　機能　 : ホムンクルスを目的地に移動させる

3) Attack (id1, id2)
　id1　: ホムンクルスの id
　id2　: 攻撃対象の id
　戻り値 : なし
　機能　: ホムンクルスに id2 を攻撃させる

4) GetV (V_***, id) end
　V_*** : 対象の情報を表す定数値
　id　: 対象の id
　戻り値 : V_***によって異なる
　　※例えば、V_POSITION の場合は x、y 座標、V_HP の場合は HP
　機能　: id の情報(V_***)を得る。情報を表す定数値は Util.lua に定義されている
　　※情報に対する詳細は「4 - ラグナロクオンラインクライアント内部定数値説明」を参照

5) GetActors ()
　戻り値 : id を返還 (LUA の table 形式に返還される)
　機能　: キャラクターの視界内のキャラクター、NPC、モンスター、アイテム、スキルなどの id を取得する

6) GetTick ()
　戻り値 : 1/1000 秒単位の数字
　機能　: コンピューターの現在の時間を取得。この値は、コンピューター起動時に 0 からカウントし、1/1000 秒ごとに 1 ずつの増加する

7) GetMsg (id)
　id　: ホムンクルスの id
　戻り値 : RO クライアントから伝達したメッセージ (LUA の table 形式に返還される)
　機能　: 使用者の直接的な命令等をスクリプトで伝達する

8) GetResMsg (id)
　id　: ホムンクルスの id
　戻り値 : RO クライアントから伝達した予約メッセージ (LUA の table 形式に返還される)
　機能　: 使用者の直接的な予約命令等をスクリプトで伝達する

10) SkillObject (id, level, skill, target)
　id　: ホムンクルスの id

level　：レベル（skill に対応）
skill　：スキルの id
target：対象の id
戻り値：なし
機能　：ホムンクルスが、対象（target）に対して level 値に対応したスキル（skill）を使用

11) SkillGround（id、level、skill、x、y）
id　　：ホムンクルスの id
level　：レベル（skill に対応）
skill　：スキルの id
x　　：x 座標
y　　：y 座標
戻り値：なし
機能　：ホムンクルスが、x、y 座標に対して level 値に対応したスキル（skill）を使用

13) IsMonster（id）
id　　：対象の id
戻り値：id にあたるのがモンスターなら 1 を、そうでなければ 0 を返還する。
機能　：モンスターを判別する

14) TraceAI（string）
string：TraceAI.txt ファイルに記録される内容。文字列
戻り値：なし
機能　：実行中のスクリプトの現在状態を記録する

4 - ラグナロクオンラインクライアント内部定数値解説

内部定数値は Const.lua に定義されています。

4-1. GetV 関数に使われる定数値

```
V_OWNER          =    0          -- 召喚者の ID
V_POSITION       =    1          -- 座標（x、y）
V_TYPE           =    2          -- タイプ（未実装）
V_MOTION         =    3          -- 現在の命令
V_ATTACKRANGE    =    4          -- 物理攻撃範囲（未実装。現在は 1 セルに固定）
```

V_TARGET	=	5	-- 攻撃、スキル使用対象の ID
V_SKILLATTACKRANGE	=	6	-- スキル攻撃範囲（未実装）
V_HOMUNTYPE	=	7	-- ホムンクルスの種類
V_HP	=	8	-- HP （ホムンクルスと召喚者）
V_SP	=	9	-- SP （ホムンクルスと召喚者）
V_MAXHP	=	10	-- 最大 HP （ホムンクルスと召喚者）
V_MAXSP	=	11	-- 最大 HP （ホムンクルスと召喚者）

4-2. GetV (V_MOTION、id) に対する戻り値

MOTION_STAND	=	0	: 立っている
MOTION_MOVE	=	1	: 移動中
MOTION_ATTACK	=	2	: 攻撃中
MOTION_DEAD	=	3	: 戦闘不能
MOTION_ATTACK2	=	9	: 攻撃する

4-3. GetV (V_HOMUNTYPE、id) に対する戻り値

LIF	=	1	: リーフ
AMISTR	=	2	: アミストル
FILIR	=	3	: フィーリル
VANILMIRTH	=	4	: バニルミルト
LIF_H	=	5	: リーフ（進化）
AMISTR_H	=	6	: アミストル（進化）
FILIR_H	=	7	: フィーリル（進化）
VANILMIRTH_H	=	8	: バニルミルト（進化）

4-4. GetMsg (id)、GetResMsg (id) によって返還されるテーブルの構造

NOME_CMD	=	0	-- 命令なし

{命令番号}

MOVE_CMD	=	1	-- 移動

{命令番号、X 座標、Y 座標}

STOP_CMD	=	2	-- 停止

{命令番号}

ATTACT_OBJET_CMD = 3 -- 攻撃
{命令番号、目標 ID}

ATTACK_AREA_CMD = 4 -- 設置型攻撃
{命令番号、X 座標、Y 座標}

PATROL_CMD = 5 -- パトロール
{命令番号、X 座標、Y 座標}

HOLD_CMD = 6 -- 待機
{命令番号}

SKILL_OBJECT_CMD = 7 -- スキル使用
{命令番号、選択レベル、種類、目標 ID}

SKILL_AREA_CMD = 8 -- 設置型スキル使用
{命令番号、選択レベル、種類、X 座標、Y 座標}

FOLLOW_CMD = 9 -- 召喚者に付いていく
{命令番号}

5 - デフォルトで提供するスクリプトの説明

5-1. AI スクリプトの必須要素

スクリプトファイルは一般的なテキストファイルです。メモ帳のような一般的なテキスト編集ソフトで作成できます。ただし、LUA という言語で書かれているため、拡張子に lua を使用しています。

ホムンクルス AI の必須条件は AI.lua ファイルと AI.lua ファイル内に定義された AI (id) 関数です。この 2 つの条件以外は全て選択事項です。
AI フォルダ内にある既存 AI.lua を他の場所に移動、もしくは名前を変えるなどして保存し、新規テキストファイルを作成してファイル名を（拡張子を含め）AI.lua に変更します。
空のテキストファイルに、

```
function        AI (myid)

end
```

と書き上書き保存します。
この状態で RO クライアントを起動すれば、正常に作動します。ただし、ホムンクルスは何も行動しません。

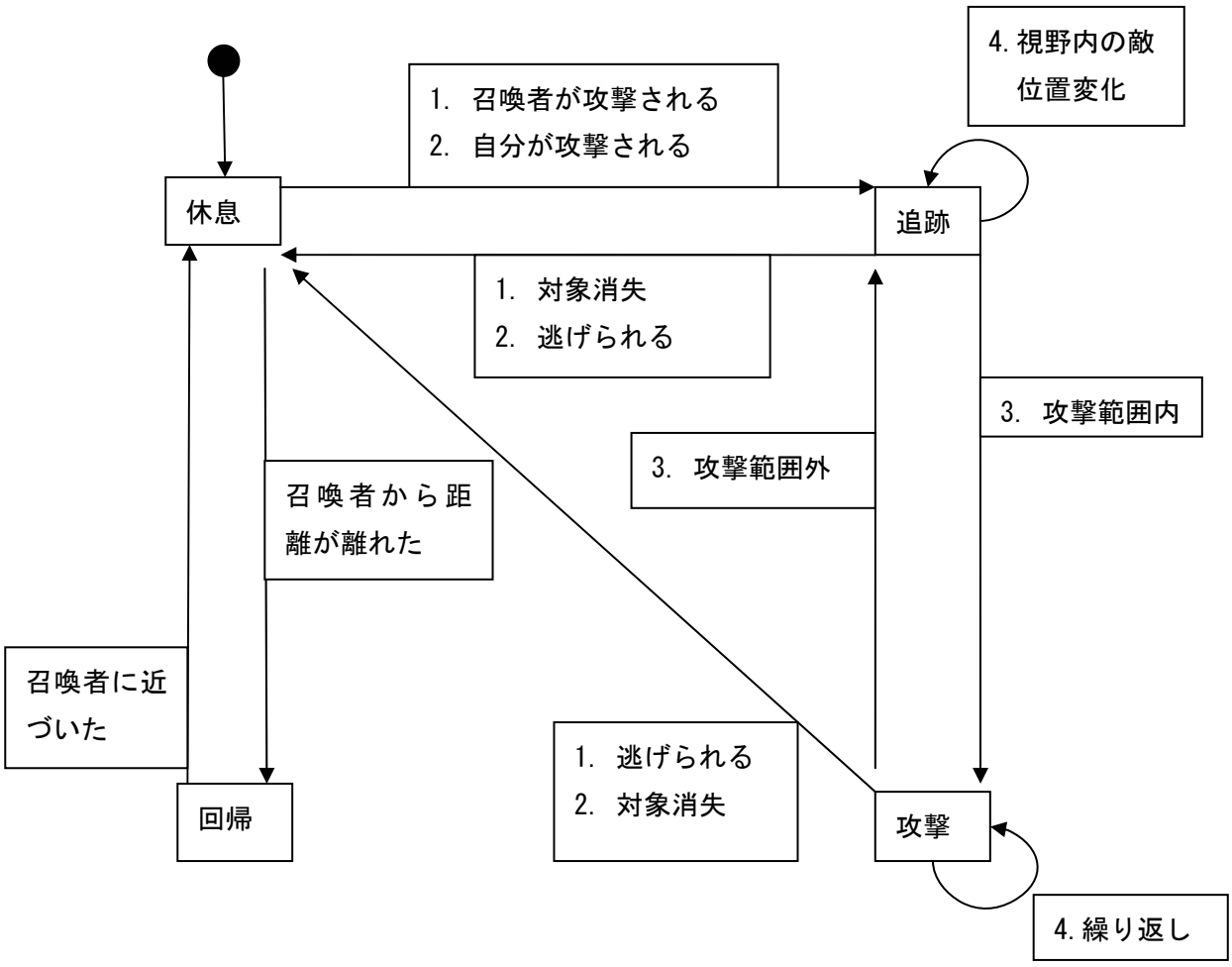
5-2. 有限状態遷移機械

AI を作成する方法はさまざまありますが、その中でも有限状態遷移機械（FSM、Finite State Machine）が、簡単で一般的に広く使われています。

では、ホムンクルスの場合は、どのような行動をさせたいでしょうか。

・周囲にモンスターがおらず、召喚者が何もしていない場合、ホムンクルスも停止する
・召喚者やホムンクルスが攻撃されたら、攻撃してきた対象を敵と認識し攻撃する
・攻撃可能な範囲に対象がいれば、物理攻撃やスキル攻撃をする
・対象との距離が、一定距離以上離れてしまった場合、召喚者の近くに戻る
等々

上記の 4 種類の状態をそれぞれ休息（IDLE)、追跡（CHASE)、攻撃（ATTACK)、回帰（FOLLOW）状態と任意に決めます。そして各状態で、どのような変化があった場合に他の状態に切り替わるのかを決めます。これらを図にすると、以下のようになります。



最初の状態は休息状態です。状況によって、ホムンクルスの状態は変化します。そしてその時々に適切な行動をするように設定すれば良いのです。さて、行動が決まったので実際にスクリプトを作成してみましょう。

5-3. Util.lua

ホムンクルスの AI を作るには、必須要素である AI.lua と AI(id) 関数だけがあれば良いですが、あらかじめ何種類かのテーブルや関数を作っておいた方が便利でしょう。これらを Util.lua に作成して、AI.lua で参照するようにします。

Const.lua を参照させるには、空のファイルの一番上に

```
require "./AI/Const.lua"
```

と書きます。

また、データを順に保存して順に取り出すための資料構造が必要な場合があります。ホムンクルスの AI の場合は、予約命令語を順に保存して取り出してくる必要があります。資料構造の名前を「 List」 ということにして、いくつかの関数を新しく追加します。

★Util.lua 参照

```
List.new ()                          -- 新しいリストを返す
List.pushleft (list, value)          -- list の左側に要素追加
List.pushright (list, value)         -- list の右側に要素追加
List.popleft (list)                  -- list の左側一番目の値を引き出す
List.popright (list)                 -- list の右側一番目の値を引き出す
List.clear (list)                    -- list を空にする
List.size (list)                     -- list に入っている要素の個数
```

よく使われる計算関数も追加します。

```
GetDistance (x1, y1, x2, y2)         -- 2 つの座標間のセル距離（定数値）
GetDistance2 (id1、id2)              -- 2 つの物体間のセル距離（定数値）
GetOwnerPosition (id)                -- 召喚者の位置
GetDistanceFromOwner (id)            -- 召喚者との距離
IsOutOfSight (id1、id2)              -- id1 と id2 がお互いに見える距離なら true、
                                        見えない距離なら false を返す
IsInAttackSight (id1、id2)           -- id1 の攻撃やスキル使用範囲に id2 がいれば true、
                                        そうでなければ false を返す
```

5-4. AI.lua

それでは、AI.lua を作成してみます。
空のファイルの一番上に Const.lua 、Util.lua を参照するように

```
require  "./AI/Const.lua"
require  "./AI/Util.lua"
```

と書きます。

次に、

```
function  AI (myid)

end
```

と書きます。

これで１つの AI が完成しました。しかし、これだけではホムンクルスは停止したまま行動しません。
先ほどの４種類の状態を定義します。

```
IDLE_ST          = 0 --- 休息
FOLLOW_S         = 1 --- 回帰
CHASE_ST         = 2 --- 追跡
ATTACK_S         = 3 --- 攻撃
```

そして現在のホムンクルスの状態を認識する変数が必要です。またホムンクルスの id、対象の id、目的地座標等も認識する必要があります。ホムンクルスの最初の状態は休息状態です。

```
-----------------------------------------
-- global variable
-----------------------------------------
MyState                  = IDLE_ST       -- 最初の状態は休息
MyEnemy                  = 0             -- 敵 id
MyDestX                  = 0             -- 目的地 x
MyDestY                  = 0             -- 目的地 y
MyPatrolX                = 0             -- 偵察目的地 x
MyPatrolY                = 0             -- 偵察目的地 y
ResCmdList               = List.new()    -- 予約命令語リスト
MyID                     = 0             -- ホムンクルス id
MySkill                  = 0             -- ホムンクルスのスキル
MySkillLevel             = 0             -- ホムンクルスのスキルレベル
-----------------------------------------

function  AI (myid)
```

```
        MyID = myid

end
```

休息状態を処理する関数 OnIDLE_ST() を定義します。そして AI(myid) 関数内に作動するように編集します。

```
function OnIDLE_ST ()
        local    object = GetOwnerEnemy (MyID)      -- 召喚者の敵を捜し出す
        if (object ~= 0) then                       -- 召喚者の敵はいるか？
                MyState = CHASE_ST                  -- 召喚者の敵を追跡状態に転換
                MyEnemy = object                    -- 敵 id を記憶
                return                              -- 関数終了
        end

        object = GetMyEnemy (MyID)                  -- 自分の敵を捜し出す
        if (object ~= 0) then                       -- 自分の敵はいるか？
                MyState = CHASE_ST                  -- 自分の敵を追跡状態に転換
                MyEnemy = object                    -- 敵 id を記憶
                return                              -- 関数終了
        end

        local distance = GetDistanceFromOwner(MyID)         -- 召喚者との距離を取得
        if ( distance > 3 or distance == -1) then           -- 召喚者との距離が 3 セル以上、
                                                               または召喚者の位置が画面内にいない
                MyState = FOLLOW_ST                         -- 回帰状態に転換
                return                                      -- 関数終了
        end
end


function AI (myid)
        MyID = myid
        if (MyState == IDLE_ST) then
                OnIDLE_ST ()
        end
end
```

召喚者が攻撃されている、または自分の敵がいる場合は、追跡状態に転換します。この 2 つともに該当しない場合は、召喚者との距離を計算して、一定距離以上離れていれば回帰状態に転換します。では、「召喚者が攻撃されている」、「自分に敵がいる」という状態はどのようにすれば分かるのでしょうか？

「召喚者が攻撃されている」という状態は、周囲にモンスターがおり、その攻撃対象が召喚者である場合です。では、モンスターではなくプレイヤーキャラクターが、召喚者を攻撃している場合はどのようにすれば分かるのでしょうか？ そのキャラクターの動作を確認すると分かります。

```
function GetOwnerEnemy (myid)                          -- 召喚者の敵を捜し出す
        local result = 0
        local owner  = GetV (V_OWNER、myid)            -- 召喚者
        local actors = GetActors ()                    -- 召喚者の視界の中の物体
        local enemys = {}                              -- 召喚者の敵を入れるテーブル
        local index = 1
        local target
        for i、v in ipairs(actors) do                  -- 召喚者の視界の中の物体を
                                                          全て確認する
                if (v ~= owner and v ~= myid) then     -- 物体の目標
                        target = GetV (V_TARGET、v)
                        if (target == owner) then              -- 目標が召喚者なら
                                if (IsMonster(v) == 1) then    -- 物体がモンスターなら
                                        enemys[index] = v      -- 召喚者の敵として追加
                                        index = index+1
                                else                           -- 物体がモンスターではないなら
                                    local motion = GetV(V_MOTION、i)    -- 物体の現在の状態
                                    if (motion == MOTION_ATTACK or motion == MOTION_ATTACK2) then
                                                                        -- 攻撃状態なら
                                            enemys[index] = v    -- 召喚者の敵として追加
                                            index = index+1
                                        end
                                end
                        end
                end
        end
```

```
	local min_dis = 100				-- 最小距離を 100 で設定
	local dis
	for i、v in ipairs(enemys) do			-- 召喚者の敵を全て確認
		dis = GetDistance2 (myid、v)		-- 召喚者と敵との距離
		if (dis < min_dis) then			-- 距離が最小距離より小さければ
			result = v			-- 一番近い敵の id を設定
			min_dis = dis			-- 最小距離を設定
		end
	end

	return result					-- 最終結果返還（ 0 なら召喚者の敵はいない）
end
```

結果、値が 0 でない場合、召喚者は攻撃を受けているということになります。

では、「自分に敵がいる」という条件はどのように判断すれば良いでしょうか？ もしホムンクルスがモンスターに先制攻撃するようにしたければ、周囲にモンスターがいたら敵がいると判断すれば良いのです。ホムンクルスが攻撃される場合にだけホムンクルスが反撃するようにしたければ、ホムンクルスを攻撃する対象があるか確認すれば良いのです。この 2 つの中から 1 つを選択する先攻、または非先攻ホムンクルスを決めます。

```
function GetMyEnemy (myid)					-- 自分の敵を捜す
	local result = 0

	local type = GetV (V_HOMUNTYPE、myid)			-- 自分はどのホムンクルスか
	if (type == LIF or type == LIF_H or type == AMSTR or type == AMSTR_H) then
		result = GetMyEnemyA (myid)			-- 非先攻型敵確認関数
	elseif (type == FILIR or type == FILIR_H or type == VANILMIRTH or type == VANILMIRTH_H) then
		result = GetMyEnemyB (myid)			-- 先攻型敵確認関数
	end
	return result
end
```

```
--  非先攻型 GetMyEnemy
-----------------------------------------
function GetMyEnemyA (myid)
        local result = 0
        local owner  = GetV (V_OWNER、myid)
        local actors = GetActors ()
        local enemys = {}
        local index = 1
        local target
        for i、v in ipairs(actors) do
                if (v ~= owner and v ~= myid) then
                        target = GetV (V_TARGET、v)
                        if (target == myid) then
                                enemys[index] = v        -- 自分を攻撃する物体を自分の敵に設定
                                index = index+1
                        end
                end
        end

        local min_dis = 100
        local dis
        for i、v in ipairs(enemys) do
                dis = GetDistance2 (myid、v)
                if (dis < min_dis) then
                        result = v
                        min_dis = dis
                end
        end

        return result
end

-----------------------------------------
--  先攻型 GetMyEnemy
-----------------------------------------
function GetMyEnemyB (myid)
```

```
	local result = 0
	local owner  = GetV (V_OWNER、myid)
	local actors = GetActors ()
	local enemys = {}
	local index = 1
	local type
	for i、v in ipairs(actors) do
		if (v ~= owner and v ~= myid) then
			if (1 == IsMonster(v))   then
				enemys[index] = v		-- 周辺のモンスターを自分の敵に設定
				index = index+1
			end
		end
	end

	local min_dis = 100
	local dis
	for i、v in ipairs(enemys) do
		dis = GetDistance2 (myid、v)
		if (dis < min_dis) then
			result = v
			min_dis = dis
		end
	end

	return result
end
```

上のように休息状態を処理するスクリプトを作成しました。休息状態で転換される追跡状態、回帰状態、また他の状態も、同じような方式で作成することができます。

最後に、ホムンクルスが使用者の直接的な命令を遂行するようにします。マウスで特定位置に移動させる、特定モンスターを攻撃させる、キーボードの特殊キーを押してその場所で待機しているようにする、といった処理のことです。

プレイヤーから降りてくる命令は、メッセージ形態でスクリプトに伝達します。そのメッセージを認識して、特定命令を遂行する状態に転換させてしまえば良いのです。その場合は、メッセージを受けて認識する部分と特定命令を処理する状態を追加して、状態処理関数を追加しましょう。AI(myid) 導入部に次を追加します。

```
MyID = myid
local msg       = GetMsg (myid)                 -- 命令を受ける
local rmsg      = GetResMsg (myid)              -- 予約命令を受ける

ProcessCommand (msg)                            -- 命令処理

-- 予約命令語保存
if msg[1] == NONE_CMD then
        if rmsg[1] ~= NONE_CMD then
                if List.size(ResCmdList) < 10 then        -- 予約命令は 10 個だけ受ける
                        List.pushright (ResCmdList、rmsg)
                end
        end
else
        List.clear (ResCmdList)     -- 新しい命令が入ってくる度に既存予約命令は取消
end
```

メッセージを処理する ProcessCommand (msg) は、各メッセージ処理関数 OnMOVE_CMD (msg[2]、msg[3]) などを追加して、それぞれの命令遂行状態処理関数（例えば OnMOVE_CMD_ST ()）なども作成します。命令遂行状態処理関数が必要な理由は、命令遂行が完了するまで命令遂行完了を確認するためです。

5-5. 文法ミスのチェックと TraceAI.txt の作成

完成したスクリプトファイルは文法的なミスを含んでいる可能性があります。文法ミスがあるスクリプトは、スクリプトがクライアントプログラムで認識される時にエラーメッセージを出します。

文法ミスがない場合でも、ホムンクルスがいる状態でゲームを始めた場合に、エラーメッセージが出る可能性があります。この時も同じく文法ミスがある部分を修正して再度ゲームを起動してください。

文法ミスがなく、スクリプトが実行されてホムンクルスが作動したとしても、論理的なミスがあり、想定通

りに作動しないこともあります。このような場合は、ホムンクルスの状態変化と各種数値の変化を記録して分析する必要があります。

例えば、 ホムンクルスが敵を追跡し、接近したにも関わらず攻撃しなかった場合は、追跡状態で何らかの論理ミスが発生していると推測できます。

```
function OnCHASE_ST ()

	TraceAI ("OnCHASE_ST")

	if (true == IsOutOfSight(MyID、MyEnemy)) then	-- 自分から敵が見えない位置にいる場合
		MyState = IDLE_ST
		MyEnemy = 0
		MyDestX、MyDestY = 0、0
		TraceAI ("CHASE_ST -> IDLE_ST : ENEMY_OUTSIGHT_IN")
		return
	end
	if (true == IsInAttackSight(MyID、MyEnemy)) then  -- 自分が敵に攻撃できる位置にいる場合
		MyState = ATTACK_ST
		TraceAI ("CHASE_ST -> ATTACK_ST : ENEMY_INATTACKSIGHT_IN")
		return
	end

	local x、y = GetV (V_POSITION、MyEnemy)
	if (MyDestX ~= x or MyDestY ~= y) then		-- 敵位置が自分の目的の位置でない場合
		MyDestX、MyDestY = GetV (V_POSITION、MyEnemy);
		Move (MyID、MyDestX、MyDestY)
		TraceAI ("CHASE_ST -> CHASE_ST : DESTCHANGED_IN")
		return
	end

	TraceAI (string.format("OnChase_ST end MyEnemy: %d、EnemyX : %d、EnemyY:%d、MyDestX:%d、
MyDestY:%d¥n"、MyEnemy、x、y、MyDestX、MyDestY))

end
```

上のように特定状態処理関数で、確認したい内容に TraceAI を入れていきます。そしてゲームチャットウィンドウに「 /traceai 」と入力すれば、RO が設置されたフォルダにある TraceAI.txt ファイルにその内容が記録されます。再び「 /traceai 」入力すれば、記録をやめます。いろいろな変数の値を記録したい時は string.format を利用して文字列を操作します。

- ver.1.0　2006 年 3 月 14 日