artisoc 上での群集流動 MAS を外部プログラムで効率化することの考察

東京大学教養学部理科一類 冨田 寛

2010 年 10 月 25 日

概要

筆者は artisoc を他のプログラムで補助することによってこそ、その真価を発揮させられると考えている。本稿では artisoc の歩行者エージェントを複数の空間で活動させ、その結果を分析するとき、外部プログラムを用いることでいかに効率が上昇するか検討する。実際に筆者が作成したドア通過モデルを例にとり、その制作過程でいかなるプログラムをもちいてルーティンワークを短縮し、効率を上げ、ミスを減らしたかを紹介する。今後更に、本考察の結果を実際の研究に適用し、実践を経て改善点を探してゆく必要があると考えている。

1 はじめに

ドア通過モデルは同じ歩行エージェントがさまざまな形状のドアを通り抜けるとき、どのようなドアを速やかに通過できるか、通過効率に大きな影響を及ぼすのはどのようなパラメータかを分析するモデルである。このモデルには様々なドアとその周囲をあらわす block 配列と、エージェントがどの部分を通過しやすいかをあらわす hesi 配列が必要である[*1]。ドアまわりは 8 種類用意し、さらに一部のマップについては複数の hesi 配列の組み合わせを作成した。研究ではこれらの組み合わせを artisoc が理解できるコードの形（これを本稿では配列の初期化用コードとよぶ）にし、背景画像をつくって artisoc 上で実行し、結果を分析しなければならなかった。この処理をすべて手作業で行うのは時間と労力の浪費である。そこで本稿では 3 種のプログラムを用いて繰り返し行う手順を簡略化した。

artisoc のポリシーは「プログラミング技法やプログラミング言語の知識は不要」な身近なマルチエージェント・シミュレータでありながら「本格的な学術研究や実務のツールにも」[1, 第 0 部 人工社会をもっと身近に] なることである。筆者はこのポリシーに賛成であり、artisoc はこの目標をよく達成していると感じる。しかし artisoc 特有のプログラミング技法を要求されるケースも散見される。他のプログラミング言語になれたユーザにとってはむしろ分かりにくいおそれがある。またドア通過モデルは小規模な研究モデルであるが、それを作る上でも artisoc と通常のエディタのみではかなりの労力を要求されただろう。単体で大規模な研究プロジェクトに利用するのは困難かと予測される。

この artisoc の欠点は、本稿で検討するように、他のプログラミング言語を用いた補助プログラムを利用することで全く問題ないレベルに解消できる。artisoc はマルチエージェント・シミュレーションの簡単かつ優れたツールと位置づけ、artisoc 内で他のプログラミング言語でならば簡

[*1] 本稿ではドアとその周囲の形状のことをマップ (map) と呼ぶ。hesi の値の分布についてもマップに含める場合もある。ドア通過モデルの詳細については 4 章–補助プログラムの実用例を参照。

単に実装できる機能を「再発明」するのをさけ、適切に外部プログラムを利用するべきである。これは手間を減らすのみならず、artisoc より高速に処理できるプログラムを並列に実行することで時間の節約にもつながる。

本稿では筆者が実際にドア通過モデルを研究する上で作成した artisoc 補助プログラムを紹介し、外部プログラムが artisoc を利用した研究にどれほど寄与できるかの実例を示したい。これらのプログラムを作成するうえで、筆者は artisoc における「プログラミングの知識を前提としない」というポリシーを受け継ぎ、プロジェクト関係者のひとりが作成すれば誰もが使えるよう心がけた。またプログラミング経験が浅い人でも比較的使いやすい HTML、JavaScript などの言語を中心に採用した。

2　artisoc 開発補助プログラム 1 —MapEditor

MapEditor は artisoc のソースコードに張り付けられる形のマップ情報と、視覚的で編集が容易なドット絵状のマップビューを相互変換するプログラムである。

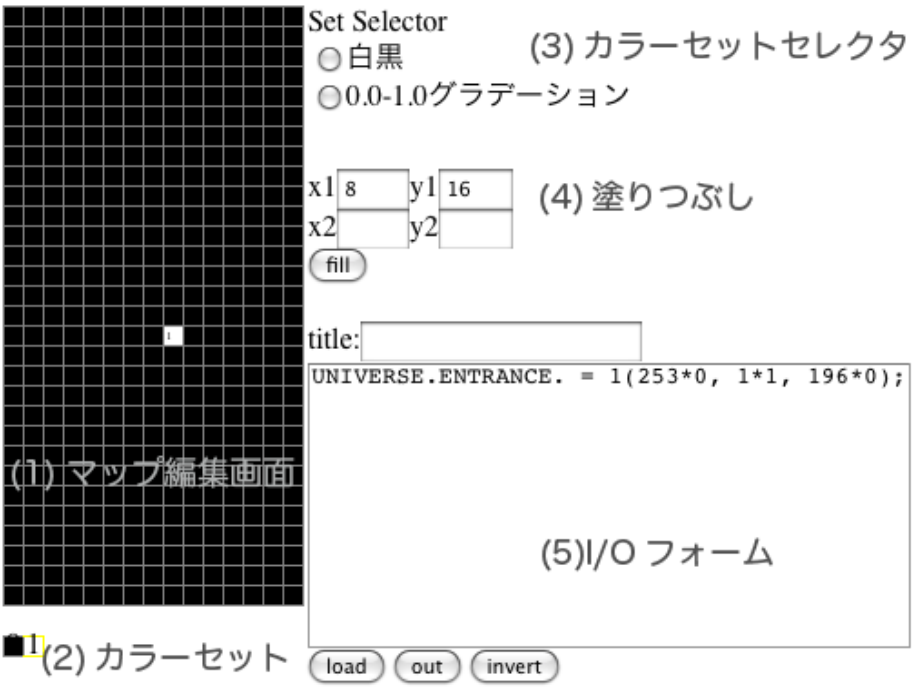

図 1　MapEditor の外観

2.1　開発動機

MapEditor はドア通過モデルを利用してどのように作られたドアが効率よく人の流れを処理できるかを検討するときに、できるだけ多くのドアの形状を再現したいという要求があった。artisoc academic において、静的なフィールドを準備するとき最も簡単で応用性が高い方法は配列を使うことである。たとえば 50x50 の空間に様々な種類の障害物などを置きたい場合、50x50 の大きさの二次元配列を用意し、空間の座標と配列の点を対応させればよい（たとえばなにもなければ 0、ブロックなら 1、資源なら 2 などといったように）。同様に、二次元配列のそれぞれの値を double

型にすることで標高や密度など連続的なデータを取り扱うこともできる [1, pp.386–387 富士山のマップを作る例]。しかし artisoc にこのような配列を用意する方法は artisoc 上で直接各点のデータを入力していくか、つぎに例示するような CSV ファイルをインポートする方法しか用意されていない [1, pp.375–376]。artisoc 上で直接入力する方式は入力したデータのリユーザビリティがない。CSV ファイルを通常のテキストエディタで artisoc に適した形式で書くことは大変な労力を要するうえ、マップの一覧性もない。

artisoc にインポートするための CSV データの例

```
"Dim:1","Dim:2","block"
0,0,1
0,1,1
0,2,1
0,3,1
0,4,0
0,5,1
0,6,1
0,7,0
0,8,1
0,9,1
0,10,0
0,11,1
0,12,1
0,13,1
⋮

3,0,0
3,1,0
3,2,0
3,3,0
3,4,0
3,5,0
3,6,0
3,7,0
3,8,0
3,9,0
3,10,0
⋮
```

マップ作成作業の敷居を下げ、マップの可視性を上げるために MapEditor が開発された。このプログラムではマップの状態を常に視覚的に確認しながら、任意の大きさのマップを作ることができる。次節以降で取り上げるように MapEditor は再利用性の高い、簡単なコードで作られているため、研究のニーズにあわせて簡単にカスタマイズできる。さらに MapEditor は artisoc なしで

作業できるため、artisoc での作業やモデル実行と並行してマップ作成が可能である。MapEditor は単にマップ作成の効率を飛躍的に上昇するのみならず、時間の有効活用にもつながった。

2.2 MapEditor のしくみ

MapEditor は HTML でカンバスとフォームを形成し、jQuery を利用した JavaScript で変換やマップ管理を行っている。

2.2.1 初期化用コードの入出力

MapEditor が生成するコードはそのまま artisoc のソースコード（.model ファイルをテキストエディタで開いて編集できるコード）にペーストすることで配列を初期化できる。本稿ではこの配列を初期化するためのコードを初期化用コードと呼ぶ。

初期化用コードの文法：

UNIVERSE.[space 名].[配列名]= 1(繰り返し数*要素の値, 繰り返し数*要素の値, 繰り返し数*要素の値, ...);

初期化用コードの例:

UNIVERSE.ENTRANCE.BASE = 1(250*0, 4*1, 196*0);

変数名、要素の型やサイズといった配列についての情報は宣言部で指定されている。初期化部には配列の大きさについての情報はなく、どの値が何回連続するかが配列の終わりまで指定されている。MapEditor を利用する際にはあらかじめ artisoc 上で配列変数を作成したうえで、図 1 の (5) の title 欄にその名前を入力して初期化用コードを生成する。空間名（この例では ENTRANCE）はソースコードで決定されている。空間名はマップの大きさとセットで固定されており、これらを変更せずに多くのマップを作成したためである。しかし、このエディタをサーバなどに設置し、様々なプロジェクトで繰り返し使うような場合にはフォームから入力できるようにするべきである。これら定数については後続のカスタマイズの節を参照していただきたい。

MapEditor では MapEditor 自身や artisoc でつくった配列初期化用コードをフォームにはりつけ、load ボタンでカンバスにそのデータを表示すること、マップ編集画面で編集した内容を out ボタンで初期化用コードに変換してフォームに出力することができる。さらに、invert ボタンで現在表示しているマップを 180 度回転したマップに対応する初期化用コードを生成することもできる。

なお、マップの向きは artisoc でモデルを作成し、何も変更しない状態（原点が左下）で正しい向きになるように出力している。

2.2.2 カラーセット

同一のモデルにおいて、異なる値の体系を使用するケースはまま生じるものである。たとえばドア通過モデルではブロックの所在をあらわす 0/1 で構成された block 配列と、各エージェントのその地点での動きやすさをあらわす 0.0 から 1.0 まで 11 段階の小数をとるふたつの hesi 配列を併用している[*2]。『人工社会構築指南』（山影、2010）にみられる富士山のモデル [1, pp.385–387] では富士山の高さと陰影を表すの 2 つのマップを利用している。

[*2] 図 1 の (3) カラーセットセレクタの二つのカラーセットはこれらに対応している。図では白黒（0/1）が選択されている。

それぞれの体系をカラーセットとして値と色の対応をつけ、それらを簡単に切り替えられるようにしたのがカラーセットセレクタである。セレクタで切り替えると、図 1 の (2) カラーセットがおきかわる。ただしこのときマップ上の値は置き換わっていないので（たとえば 0/1 で編集したあと、カラーセットをグラデーションに切り替えても 0/1 が 0.0/1.0 に置換されたりはしないので)、正しいデータにするためには手作業でそれぞれを対応する値に置き換えなければならない。これは前のマップにもとづいて次のマップを編集したいというニーズと、カラーセット A のある値がカラーセット B の特定の値に必ずしも対応しないゆえの仕様である。したがってモデルを作る上でカラーセット間の値の対応がほぼ例外なくきまっている場合、カラーセット切替時に自動的に置換するようにプログラムを書きかえるとよいだろう。

なお、例ではグレースケールのカラーセットしか用意していないが、別の色のグラデーションにすることもできる。白–0 は空間、黒–1 は障害物、赤–2 は資源といったようにブロックの種類に応じてバラバラの色を使うこともできる。

2.2.3 マップの編集

マップの編集方法は、ペイントソフトのように、カラーセットで色を選び、マップをその色で変更することの繰り返しである。マップの変更の仕方には 2 種類あり、ひとつはカンバス上でマウスをクリック＆ドラッグすることでなぞったマスの色を変えていく鉛筆ツールのような操作である。もう一つは範囲を指定して塗りつぶす操作で、図 1 の (4) 塗りつぶしで始点と終点を指定して fill ボタンを押すことで、その 2 点で囲まれる範囲が塗りつぶされる。ペイントソフトでは矩形選択＋塗りつぶしツールに対応する。マップ編集領域上でマウスを押しさげたとき、その座標が塗りつぶしツールの x1 y1 に自動的に格納される。

2.3 MapEditor のカスタマイズ

MapEditor は HTML と JavaScript で構成されており、これらの言語について基礎的な知識があれば各モデルや研究のニーズに合わせてつくりかえることができる。

2.3.1 マップ情報の設定

MapEditor で指定できるマップ情報にはマップの幅、高さ、space 名がある。これらは mapEditor/gen.js 内で指定されている。

mapEditor/gen.js

```
var map_x = 15;
var map_y = 30;

var space = "ENTRANCE";
⋮
```

これらを次のように書きかえれば、幅 50、高さ 100、space 名は test となる。artisoc で入力する空間名は大文字でも小文字でもかまわないが、MapEditor の space 名として記述するときはすべて大文字で書くべきである。これは artisoc のコードでは大文字と小文字が区別されないが、一般的に初期化部では全て大文字で表記するという原則による。

mapEditor/gen.js

```
var map_x = 50;
var map_y = 100;

var space = "TEST";
⋮
```

2.3.2 カラーセットの設定

カラーセットは次の手順で定義する。

1. カラーセット用配列を宣言する。
2. カラーセット用配列を数値をキー、カラーコードを値とした連想配列にする。
3. 連想配列 ccode にカラーセット配列名をキー、カラーセット配列の持つ数値のリストの配列を値として追加する。
4. HTML ファイルにカラーセットセレクタを追加する。

ただし、2 番目と 3 番目の手順は逆でもよい。mapEditor/gen.js には次の例がある。

mapEditor/gen.js

```
grad = new Array(); //Step. 1

ccodes['grad'] = ["0.0","0.1","0.2","0.3","0.4","0.5",
                  "0.6","0.7","0.8","0.9","1.0"]; // Step. 2

grad['0.0']='#000'; // step. 3
grad['0.1']='#111';
grad['0.2']='#222';
grad['0.3']='#333';
grad['0.4']='#444';
grad['0.5']='#555';
grad['0.6']='#666';
grad['0.7']='#777';
grad['0.8']='#888';
grad['0.9']='#999';
grad['1.0']='#AAA';
```

この連続的な値の設定をプログラムで行うなどの工夫もできる。

つぎに HTML ファイルを編集し、カラーセットセレクタを追加する。40 行目付近に次のような部分がある。'mono'、'grad' の部分が配列名に、「白黒」などの部分がそのカラーセットの説明である。新しく作ったカラーセットをここでセットできるようにすれば、ラジオボタンをクリックしたときにそのカラーセットがロードされるようになる。

mapEditor/mapEditor.html

```
<div style="padding-bottom:30px">
Set Selector<br>
<input type="radio" name="choice" id="choice" onclick="startup('mono')">
白黒<br>
<input type="radio" name="choice" id="choice" onclick="startup('grad')">
0.0-1.0 グラデーション
</div>
```

2.3.3 ソース解説

gen.js の概略を説明する。HTML コードはたんにフォームを形成し、JavaScript の関数を呼び出しているだけなので割愛する。

startup
: カラーセット名を引数にとり、環境のセットアップを行う。初回ロード時には map_x, map_y にしたがってカンバスを作成し、カラーセットの中身を展開してパレットに割りつける。二回目からはパレットの更新のみ行う。

load
: load ボタンで呼び出される。フォームに張り付けられた初期化用コードを正規表現などをつかって解釈し、カンバスの色と値を付けかえる。

out
: out ボタンで呼び出される、出力用関数。カンバスの状態を取得し、上記初期化用コードのフォーマットに合わせて出力する。

fill
: fill するときに実行される関数。始点と終点を取得し、塗りつぶす。

3 artisoc 開発補助プログラム２—SubModelGenerator

本稿ではテストの対象となる個々のモデルをサブモデル (SubModel) と呼んでいる。ドア通過モデル全体で同じエージェントルールを使うが、個々のサブモデルでは空間の大きさとマップが異なっている。この個々のサブモデルを MapEditor の編集結果を用いて半自動的に作成するためのプログラムが SubModelGenerator である。

3.1 開発動機

ドア通過モデルを構築するにあたって 10 種類以上のサブモデルを作成する必要があった。これら多量のモデルを管理する最も簡単な方法は、それぞれのサブモデルに必要な 3 種類のマップの配列初期化コードをテキスト形式で用意し、エージェントルールを含む.model ファイルの雛形に埋めこむことであった。この作業を手作業で行なう場合、マップの大きさが様々なところに埋め込まれているため変更の手間がかかることや、マップの情報を反映してブロックのありなし、hesi 変

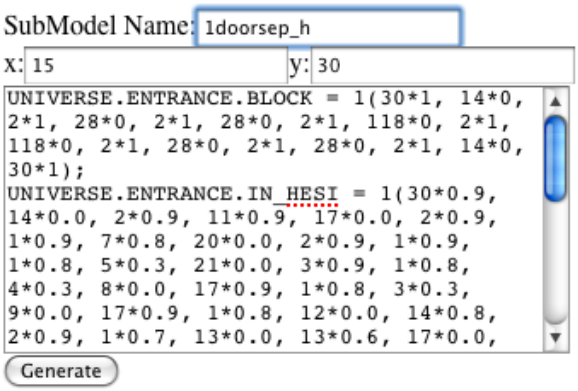

図 2　SubModelGenerator の外観

数[*3]の大小をしめした背景ファイル（gif ファイル）を用意しなければいけないといった問題点がある。

SubModelGenerator はマップ初期化コードの保存、.model ファイルの作成、gif 背景画像の生成を自動で行ない、一式のファイルを zip 圧縮して提供する。このプログラムは MapEditor よりもドア通過モデルと密に関係しており、応用性は比較的低いかもしれない。しかし、png ファイルの作成処理・おなじエージェントルールをもったたくさんの.model ファイルの作成管理などは他のプロジェクトに部分的にでも応用できよう。

3.2　SubModelGenerator のしくみ

SubModelGenerator は HTML と php で構成されている。php は画像処理用の関数群（GD サポート [2]）が利用できる必要がある。また、zip ファイルを作成するために用いている zip.lib.php は phpMyAdmin[3] の一部として提供されているものである。あらかじめ雛形となる subModelGenerator/template.model ファイルを用意する。HTML フォームからサブモデルの名前、空間の幅・高さ、マップの初期化用コードを設定し、結果出力される zip ファイルをダウンロードして利用する。

マップ情報はフォームから入力することができる。この情報は加工されて、template.model 上の特殊な変数（{, } で囲まれたもの）に代入される。たとえば幅として入力された値は php 上の変数$x に格納され、計算処理を行った上で、template.model にふくまれる {$x}、{$xdec}、{$xy} などに代入される。

3 種類のマップは初期化用コードとして雛形の中に埋め込まれ（例では {$DAT} とおきかわる）、map.txt という名前で保存される。同時に変数名からどの行がどのマップに対応するかを判断され、それぞれ artisoc 内とおなじくニ次元配列に展開される。各座標に対応する点の値にしたがって 1 ピクセルずつ色付けされ gif 形式で背景画像として出力される。背景はモデルが制作者の意図通りに実行されていることの視覚的確認や非プロジェクト参加者の理解促進に役立つ。背景画像を外部のファイルに依存し固定することによって、描画処理を軽量化し、モデルの実行を高速化する狙いもある。

変数に値を代入された雛形は [サブモデルの名前].model というファイルとして保存される。以上 3 つのファイル—.model ファイル、map.txt、back.gif—が含まれた zip ファイルが生成され、

[*3] エージェントの進みにくさを表す変数。詳細は 4.2.2 節–エージェントのルール。

サブモデルの名前を冠した zip ファイルとしてローカルに保存される。サブモデルをテストするためにはこの zip ファイルを解凍し、なかの model ファイルを開いて実行するだけでよい。

3.3 SubModelGenerator のカスタマイズ

SubModelGenerator の初期化部は次のようになっている。

$space　空間の名前を設定する。artisoc と MapEditor における名前と同じでなければいけない。

$bgname　背景画像の名前を設定する。artisoc での名前と同じでなければならない。拡張子は gif にすること。

$rules　置換のルールを設定する。連想配列のキーが検索クエリ、値が代入すべき文字列。代入文字列はフォームから送られてきた値から作るのが一般的だろう。

$mapnames　読み込むべきマップ名を設定する。それぞれのマップの値に即してどのような色付けをするかは、画像出力部分付近で指定されている。

subModelGenerator/download.php

```
⋮

$space = 'ENTRANCE';
$bgname = 'back.gif';

$x = intval($_POST['x']);
$y = intval($_POST['y']);
$DAT = $_POST['dat'];

$modelname = ($_POST['modelname'] || $_POST['modelname'] != '') ?
  $_POST['modelname'] : 'doors';

// replacement rule
$rules = array(
               '{$x}' => $x,
               '{$y}' => $y,
               '{$DAT}' => $DAT,
               '{$xdec}' => $x-1,
               '{$ydec}' => $y-1,
               '{$xy}' => $x * $y,
               );

$mapnames = array ("BLOCK", "IN_HESI", "OUT_HESI");

⋮
```

4　補助プログラムの実用例–ドア通過モデル

本章ではドア通過モデルの実際の研究の流れにそって、前 2 章で紹介した補助プログラムがどのように生かせるかを検討する。あわせて前 2 章のものより小規模な分析用プログラムも紹介する。

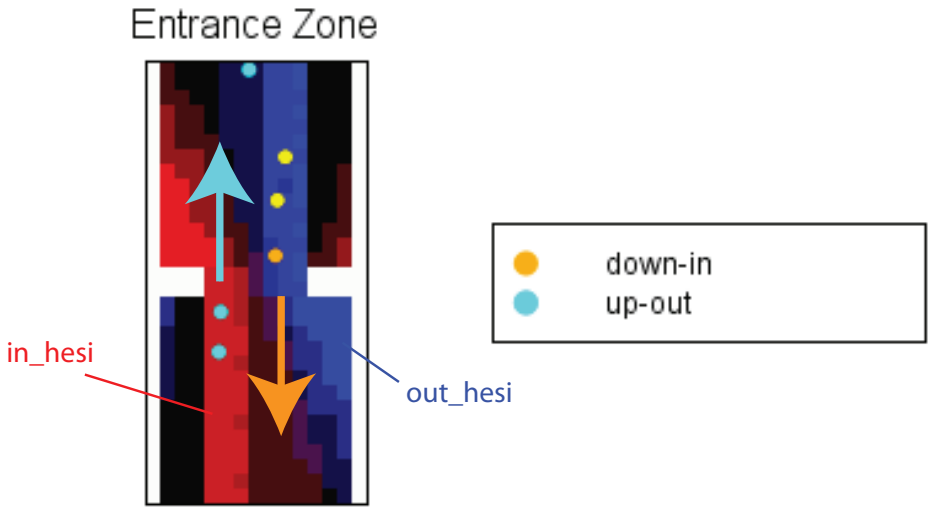

図 3　ドア通過モデルの実行画面

4.1　ドア通過モデルの目的

このモデルはわれわれが日々利用するさまざまな施設の入り口や部屋のドアの人の流れをどうしたらスムーズにできるかを検討する目的で作られた。多様な形のドアの効率のよさはどのようなパラメータによって大きな影響をうけるのかを憶測を排し、artisoc のエージェントシミュレーションによって実験的に考察した。効率のよさとは出るエージェントと入るエージェントのそれぞれが下端（上端）から上端（下端）に達するまでの平均ステップ数と考え、エージェントの密度を変化させて効率の変化を計測した。

この研究においては多くの種類のドアを考え、マップとして用意する必要があった。また、人のながれをロープや係員の誘導によって制限したとき、あるいは自然とできる人の流れにみなが従うときに効率がどう変化するかを考察するため、それぞれのエージェントの動きやすさを指定する hesi 配列も用意した。ロープや誘導が原因で通りにくいところが hesi 変数の大きな、エージェントにとって早く動きにくい場所である。これらのマップの組み合わせを迅速にシミュレートするために、今まで考察した補助プログラムを用いた。

4.2　シミュレーションルール

すべてのマップにたいして同じシミュレーションルール[*4]を使用した。これは人々はどのマップでも同じルーティンで出入口を通行しようとし、ドアの形状によらないとの推定からである。

[*4] シミュレーションルールとは、エージェントの持つルールとエージェントたちを統括する Universe のルールをあわせたもののことである。

4.2.1 Universe のルール

Universe はおもにエージェントの出入り、平均ステップ数のカウント、それらの値の出力を行う。次の変数を持つ。

universe.status　出現するエージェントの多寡を実験者が指定する。値が大きいほど、単位時間あたりに現れるエージェントが多い。実際のシミュレーションでは artisoc の連続実行機能を用いて一定回数実行するたびに増加させた。過密状況（後述）以外では 1 から 10 の整数値をとる。

universe.u_birth　u は up を意味し、出るエージェントに対応する。出るエージェントがどれだけ発生したかを記録する。

universe.u_sums　出るエージェントが下端から上端に達するまでに要したステップ数の合計を記録する。各エージェントが上端に達するたびに所要ステップ数を報告し、平均値を出力するときに birth で割って平均を出す。

universe.d_birth　d は down を意味し、入るエージェントに対応する。機能は u_birth と同様。

universe.d_sums　u_sums と同様。

値の出力先は連番のファイル（output1.txt から output100.txt）で、次の CSV 形式でかかれている。サブモデルは各回 1500 ステップ実行され、そのうち 1000 ステップ以降を十分な時間が過ぎたあとの値として記録した。この記録に対して 4.4 節—分析で紹介するプログラムと Microsoft Excel の表計算・グラフ機能を用い、移動効率の分析を行った。

output*.txt の書式

```
[回数] - status:[status の値]
現在のステップ数, 出るエージェント平均, 入るエージェント平均
現在のステップ数, 出るエージェント平均, 入るエージェント平均
現在のステップ数, 出るエージェント平均, 入るエージェント平均
⋮
```

output34.txt (実例)

```
34 - status:0.4
1000,25.3,26.533333333333335
1010,25.3,29.133333333333333
1020,28.4,29.133333333333333
1030,25.818181818181817,29.133333333333333
1040,25.818181818181817,29.133333333333333
⋮
```

4.2.2 エージェントのルール

エージェントはつぎのような原理で動いている。

- speed 変数の速さで Direction 変数の方向に移動する。
- 色は RGB 値 view 変数を反映する。エージェントのイメージカラー（Red-入るエージェント、Blue-出るエージェント）にくわえて、速度が早いほど Green の値が大きい。
- 壁にぶつかったら、壁の無い方に方向転換する。
- 壁にぶつからないときは前方にすすむ。
- 自分の前に行きたくない場所 (entrance 空間の hesi 変数が大きい場所) があったら避ける。
- 自分の進行方向、右 45 度、左 45 度を見て、仲間エージェントを探す。仲間エージェントがいたら、ぶつからないように速度を下げつつ、一番仲間が多い方に追従する。
- 仲間がおらず、相手がいたら、ぶつからないように速度を大きく下げて避けるように進む。
- 仲間も相手もいなかったら、一生懸命に前に進む。
- 誕生時には x 座標のみをランダム生成し、そこがブロックでないことを確認してから誕生する。
- 上端に達したら所要ステップを universe の sums 変数に報告し、消滅する。

入るエージェントと出るエージェントの行動原理は点対称につくられている。

4.3 マップ

次の種類のマップを用意した。これらはすべて MapEditor で描かれたものである。マウスで線を引くという簡単な操作で作成できるため、各マップを用意するのにかかった時間は数分から十数分である。図 4 から図 8 の一般的なドアのマップは hesi 変数をもつものともたないものを用意し、もつものについては universe.status の各値を 1 回ずつ、もたないものについては各値を 10 回ずつ実行した。のこりの図 9 から図 12 はドア通過モデルが様々な種類のドア空間を分析できることを示している。これらのサブモデルを作る上でもエージェントルールにはほとんど手を加えておらず、始めから実装している hesi 変数の利用と MapEditor と SubModelGenerator の活用で作成している。この事実は拡張性を考えてエージェントルールをつくり自由度の高い補助プログラムを活用することで、多様なサブモデルの比較を少ない手間かつ高い信頼性で行えることをしめしている。

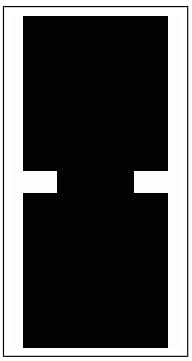

図 4　通常マップ

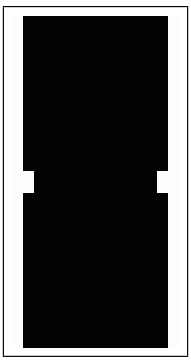

図 5　大ドアマップ

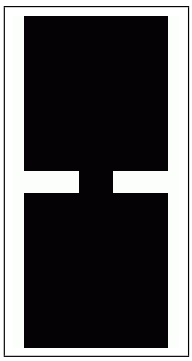

図 6　小ドアマップ

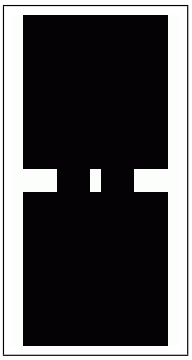

図 7　1 ドア分割マップ

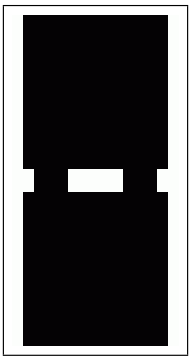

図 8　2 ドア分割マップ

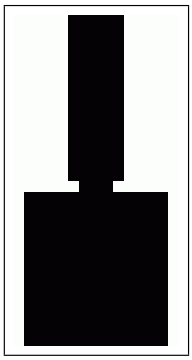

図 9　道狭マップ

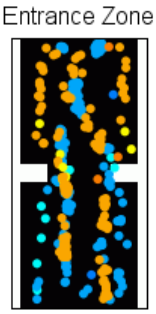

図 10　過密状況

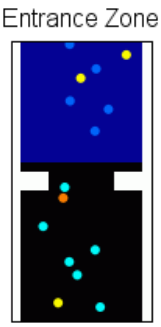

図 11　坂マップ

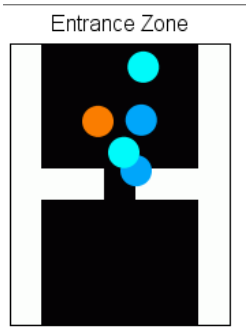

図 12　室内

4.4　分析

SubModelGenerator に MapEditor で作成した初期化用コードを入力し、10 以上の zip ファイルを得てすべてのモデルを実行した。モデルの準備は短時間で終了したが、その実行には数日を要した。モデルの実行終了を確認し、次のモデルファイルを開いて実行するようなキーボード操作プログラムを利用すべきであった。上述の output*.txt ファイルを処理し、Microsoft Excel に取り込んでグラフ化した。output*.txt ファイルを Excel に適した形に変換するために parse.cc という C++ プログラムを用いた。

4.4.1　parse.cc について

parse.cc は artisoc が出力した結果の平均を取り、Excel で扱いやすいように変換して出力する。hesi 変数がある状態 (hesi mode) とない状態 (no_hesi mode) では実行回数やファイルの名前規則が違うので、別のプログラムファイルになっている。それぞれ parse/parse_h.cc と parse/parse_nh.cc である。

いずれの状態についても universe.status 変数が 1 から 10 のそれぞれのケースについて、入るエージェントと出るエージェントのそれぞれについて所要ステップ数の平均を計算している。1000 ステップ目から 1500 ステップ目は開始から十分に時間がたっているという同様の条件にあると仮定し、その平均を計算したのち、no_hesi mode についてはおなじ universe.status で実行された 10 回の結果を平均している。次は出力ファイルの例である。

出力ファイル例：1sep_nh.txt

```
# 1sep_nh_o 1sep_nh_i
1 32.7517 31.6062
2 34.8685 33.7052
3 37.0265 35.8042
4 38.9427 37.8738
5 40.5418 39.3286
6 41.8707 40.5168
7 43.02 41.5451
8 43.8594 42.4727
9 44.55 43.0465
10 45.1313 43.6854
```

このプログラムは応用性が低く、Excel など他のソフトウェアでも行える処理しかしていない。artisoc 内で変数を利用していきなり同じ結果を出力することも可能である。しかし、加工の少ないデータを出力することはシミュレーションに負荷をかけず処理が高速、必要なデータが増えてもシミュレーションをやり直す必要がないなどの利点がある。分析プログラムをかきかえることで同じ出力ファイルが各 status 値にたいする所要ステップ数の分散を調べたり、ドアの幅を 2 マス刻みで用意して幅と所要ステップ数の相関関係を調べたりといった発展的な調査に応用できる。また、CLI アプリケーションであるためサブモデル数が多くなったときに一括処理で威力を発揮するだろう。

4.5 結果の概要

parse.cc の出力結果をいろいろな観点からまとめたグラフを列挙する。これだけ多くのモデルの比較ができることは今まで解説した補助プログラムが多くのマップの比較に威力を発揮することを示唆している。

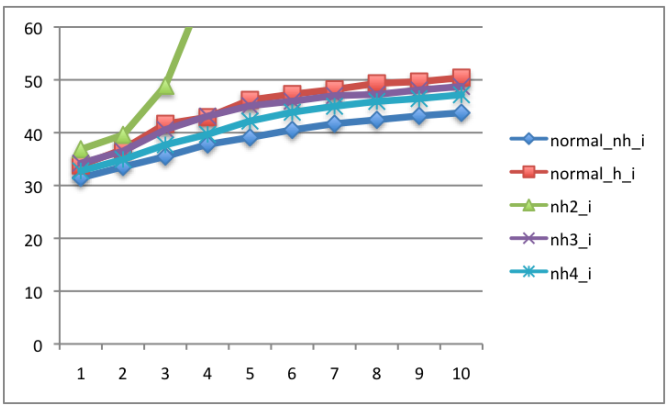

図 13　通常マップにおけるさまざまな hesi マップ間の比較。比較対象のエージェントは入るエージェントのみ。nh は no_hesi を意味し、これがもっとも効率的である。

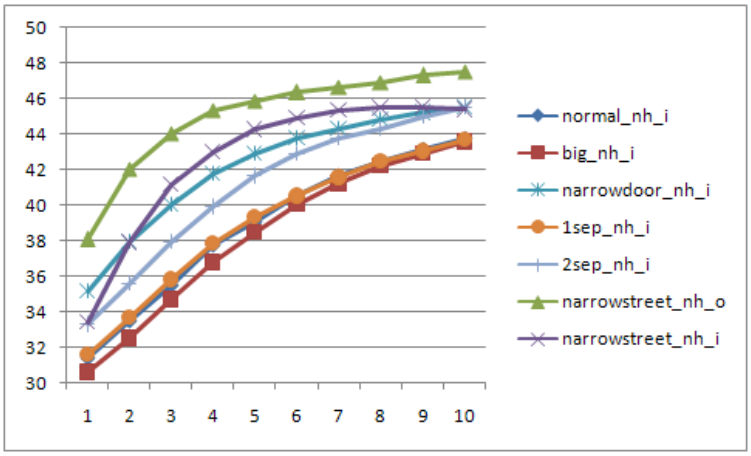

図 14　入るエージェント同士の異なるマップでの比較。13 において no_hesi mode が最も効率的であると推測されたので、すべて no_hesi mode で比較している。narrowstreet(道狭マップ) は対象ではないので in と out 両方のエージェントを含めている。

5　まとめ

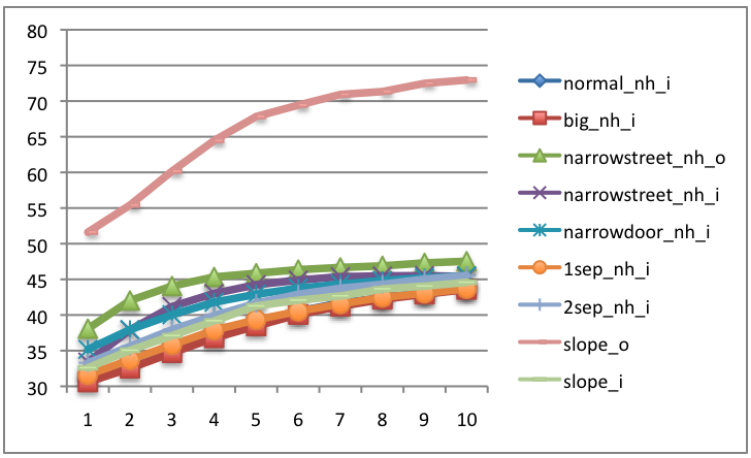

図 15　坂マップを含めた結果。坂を登るエージェントは通常の倍近く遅い。ドアの先に階段などを設置すべきではないと推測される。

以上ドア通過モデルを軸にそれをとりまく 3 つの補助プログラムの検討をおこなってきたが、ドア通過モデルもその補助プログラムもあくまで一例である。同じ動作をするプログラムでも色々な実装手段があり、同じ結果を実現するにも色々な道筋がある。さらに、ドア通過モデルのエージェントの質は高くなく、改善する必要が多いにある。だが、本稿の目的は artisoc が外部プログラムの利用によって持ち前以上の能力を発揮できることを示し、外部プログラムの一つの例を示すことであった。多くの研究現場で同じようなルーティンワークがなされ、そのために時間と労力が割かれている事態は見過ごされるべからざるものである。ひとつ補助プログラムを作成すれば、そのプロジェクトのみならず、後の研究やほかの団体による研究にも活かすことができるだろう。筆者が今回サンプルとしてあげたソースコードを公開し、若干の説明を加えたことには稚拙なコードながら、どこかでふたたび活用されえるのではないかとの願いからでもある。本稿を参考にそれぞれの研究現場で作業効率をあげる工夫がなされれば、筆者にとってこれほど嬉しいことはない。

付録 A　外部プログラムについて

筆者が作成し公開している補助プログラムは一例であるから、これを様々に改造して利用していただきたい。各章で紹介したようにある程度の拡張性を備えている。

添付のプログラムはすべて Mac OS X Leopard と Firefox 3.6、Apache2 と PHP5.3 の環境で動作確認をしている。とくに MapEditor は最近のブラウザを要求する点、jQuery のソースコードをネットを介して参照するためにインターネット接続を要求する点に注意されたい。プログラムにバグや不具合を発見した場合、使用法や改造法で不明な点がある場合などはメールにて <tomykaira@gmail.com>まで連絡されたい。くれぐれも、artisoc と直接関係ない問題について artisoc の開発に関係する構造計画研究所や山影研究室に問い合わせることがないように気をつけていただきたい。

すべての JavaScript, PHP, C++ のソースコードは GNU General Public License Version 3 のもとで公開され、著作者は TOMITA Hiroshi <tomykaira@gmail.com>である。ただし、HTML コードと SubModelGenerator に含まれる zip.lib.php は除く。zip.lib.php は phpMyAdmin[3] のソースコードに含まれるものであるため、そのライセンスに従っていただきたい。なお、SubModelGenerator に同梱している template.model は動作確認用の見本ファイルであり、ルールは記していない。

参考文献

[1] 山影進 『人工社会構築指南 artisoc によるマルチエージェント・シミュレーション入門』 改訂新版 (書籍工房早山, 2010 年)

[2] the PHP Documentation Group, PHP マニュアル翻訳プロジェクト “PHP: GD – Manual” Last updated: Fri, 01 Oct 2010 http://php.net/manual/ja/book.image.php

[3] “phpMyAdmin” http://www.phpmyadmin.net/home_page/index.php