Jasonシミュレータにおける Map拡張による実世界空間の表現

奈良女子大学 情報科学科
07251681 岸上 文

平成23年2月14日

概 要

マルチエージェントシステムとは、複数の自律したエージェントから構成されるシステムである。マルチエージェントシステムにおいては、各々のエージェントは自分の状況を知覚し、人工的な社会を構成し、相互的に作用し合う。こうしたシステムの挙動は、複雑なため実世界での試行は困難であるので、その開発においてはシミュレーション技術が必要とされる。我々はエージェント開発用ソフトウェアである Jason のシミュレーション環境のジェネレーターとして、パラメータの初期設定機能・土地拡張機能を持ったものを開発した。本論文では、これについて述べる。

1 はじめに

シミュレーションとは、実世界などの仮説状況をコンピュータ上でモデル化するもので、シミュレーションによって、そのシステムがどのように作用するのかを調べることができる。シミュレーションは、今や様々なシステムのモデル化、経済・社会学における人間に関わるシステムのモデル化、工学システム等の洞察を得る手段となっている。例として、ネットワーク交通量シミュレーションが挙げられるが、このようなシミュレーションは、その環境においての初期設定を変更すると、モデルの振る舞いは変化する仕組みになっている。

このように、シミュレーション環境を自分達で作成しシミュレーションする例はあるが、特定の空間上、または問題において膨大なデータを用いて、実際にシミュレーションを行う例は多く見られるのに対して、何もない状態からシミュレーション環境を新しく作り出す機能は提供されていない。特にマルチエージェントシステムの開発においては、開発すべきプログラムが一般的に複雑になるため、シミュレーション環境の提供が強く望まれる。しかし、マルチエージェントの開発環境として使われている Jason[1] には、そのような要請を満たすシミュレータは提供されておらず、サンプルプログラムとして、決められた大きさの空間上でのシミュレーションを行うものが添付されているに留まる。更に実際の開発においては空間の大きさやエージェント数などの設定を様々に変えてシミュレーションできることが望ましい。

そこで今回我々は、上述した Jason のサンプルプログラムをもとに、Map(エージェントの行動範囲) の広さを実行時に動的に拡張できるように改良し、実世界空間の表現に近付

けたシミュレーションを可能とした。また、初期設定において Agent の数・各々の初期位置・初期の描画領域を入力する機能を持つジェネレータを開発し、様々な環境でのシミュレーションを可能とした。

既存のマルチエージェントシミュレータとしては [3]、社会シミュレーションモデル構築用の SOARS[4](Spot Oriented Agent Role Simulator) や社会・経済のモデル化を支援する道具立てとなる Plat Box[5] などが挙げられる。SOARS は人間の意志決定を主体としており、人間の行動をルール化し、人間の行動が反映されたモデルを構築しているシミュレーションモデルで、表示方法は XY グラフで表示される。また PlatBox はシミュレーションを動かすことによって社会を理解し、複雑な社会の理解を助けるモデルを作成するもので、表示方法としてグラフやリレーションがある。これらは、社会現象をモデル化することを目的としたものであるので、新しく開発するエージェントの動作を視覚化するためのものではない。それに対し、我々のシミュレーションは、エージェントの開発時に、そのエージェントのシミュレーション環境を新しく作り出すことによって開発を助けることを目的としている。

なお、本研究は本学 4 回生の岡田 [6] との共同研究であり、本論文ではそのうちの Map 拡張の実装について述べる。

本文の構成を示す。２節では本研究の出発点となった既存シミュレーション環境とその問題点について述べ、続いて３節ではビット列を使ったオブジェクトの表現方法と Map の動的拡大の方法について述べ、４節では例題として Treasure の発見を目標とした問題を用いての検証実験について述べる。最後にまとめと今後の展望を述べる。

2　既存シミュレーション環境の問題点

2.1　サンプルプログラム

Jason には domestic-robot というサンプルプログラムが添付されている。このプログラムでは、1 体のロボット (図 1) が、owner にビールを出すという目標を持っている。ロボットは owner からビールのリクエストをいくつかもらい、冷蔵庫 (fridge) まで進みビールの瓶を取り出し、それを持って owner の所に戻る。しかし、ロボットはビールの在庫状況も気にしなければならない (最終的にはスーパーの配達サービスを使ってビールを注文することになる) といったものである。

そして、このサンプルプログラムの中に Map という描画形式のクラスが存在し、width と height で Map の固定幅を設定し、Location オブジェクトを用いて２次元の座標を表している。全ての Map を二次元配列で形成し、座標内の状態を CLEAN・AGENT・OBSTACLE で表している。この実現方法は、それぞれビット列 0,2,4 を用いて表わしている。また Agent の初期位置を Location 配列を用いて表し、初期値を $(0, 0)$ とおいていた。

また、Jason でシミュレーションする際には必ず「知覚」という動作が起こり、その知覚した内容を次々に「信念」に加えていく必要がある。ロボットはまずその状態が存在するという事を「知覚」し「信念」に加えていかない限り、自発的に次の行動を起こすことができない。一度「知覚」が得られると、信念ベースは環境の変化を反映するごとに、更

新する必要性がある。そこでサンプルプログラムでは、ロボットの知覚によって信念に変化が起こると、それを Map の描画に反映させる。

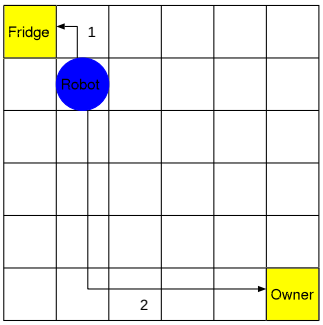

図 1: サンプルプログラムの実行の様子

2.2 問題点とその解決

このサンプルプログラムは二次元座標内で描画されており、空間の大きさが一定に固定されている。しかし、一般的には、エージェントの行動範囲があらかじめ定められた一定の大きさに制約されるとは限らない。このため、一般的なシミュレーション環境として使うには十分ではない。

そこで実世界空間により近づけるために、Map の拡張機能を付け加える環境を追加した。サンプルプログラムでは描画の際に二次元配列を使用していたが、それでは負の添字が表現できないこと、Map の拡張が実現できない (最初に要素数を設定するため、Map 拡張は実現不可能である) という問題が生じた。後者は ArrayList を使うことで解決できるが、前者はこれでは解決できない。そこで HashMap[2] オブジェクトを使って負の座標を表現することにした。また Map の描画に関しても、同じく二次元配列を使用しているので、ここも変更する必要があると判断した。次章では、HashMap オブジェクトを使った描画方法について説明する。

3 実装方法

3.1 HashMap について

HashMap とは、オブジェクトへの参照を保持するクラスである。HashMap は要素を「キー」(オブジェクトに付けた名前) を使って管理し、追加する際は、「キー」を指定して追加する (図 2)。つまり、指定された値と指定された「キー」をマップに関連付けることができる。また任意の「キー」が使えることも HashMap の利点と考えられる。また HashMap クラスは Map インターフェースを実装したクラスの中で、最も高速に動作する。以下に例を挙げる。

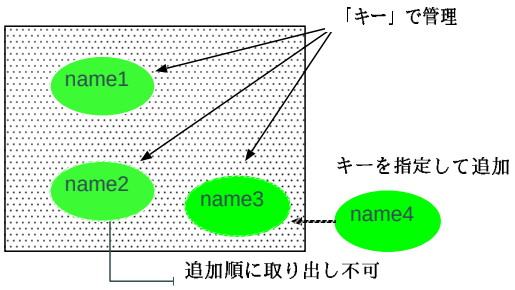

図 2: HashMap オブジェクト

- HashMap<Integer,String> map = new HashMap<Integer,String>();

HashMap の作成時には型を 2 つ指定する。1 つ目はキーの型でこの例では Integer、そして 2 つ目は格納する要素の型でこの例では String である。要素を格納する時は put メソッド、要素を取り出す時は get メソッドを使う。キーと要素の型はそれぞれ自由に決定できる。今回はこの特性を利用し、Msp 上の位置とそこにあるもののデータを対応づけるために HashMap を使用した。

3.2 描画方法

今回は、HashMap で保持されるキーの型を Location 型の (X, Y) 座標と設定し、Map される値の型を Integer 型とした。サンプルプログラムは座標内の状態を CLEAN・AGENT・OBSTACLE で表し、それぞれの存在表記をビット列 0・2・4 と表しているが、この点は我々のプログラムでも踏襲した。そしてどのオブジェクトが存在するのかをビット演算で判定し、描画することにした。例えば図 3 左のように、$(0, 0)$〜$(2, 2)$ の 3×3 のマスのうち、$(1, 1)$ に AGENT がいる状態を描画させたいとする。まず、Loction 型変数 cood に座標を入れ、model.data.get(cood) でビット列を取り出す。図の場合だと $(1, 1)$ 内には Agent がいるので、ビット列 2 が返ってくる。そして、AGENT のビット列 2 とのビット and を＆演算で求め、0 でない場合に青い円を描くようにした。また OBSTACLE の描画は座標内を黄色に塗りつぶした (同図中央)。

同じマス目に複数のオブジェクトが重なった状態を描画したい場合は以下のようにする。今、$(0, 0)$ に AGENT と OBSTACLE が重なっていたとする。その場合は、model.data.get(cood) に、AGENT の 2 と OBSTACLE の 4 を足した値 6 が格納されている。先ほどの例と同じように、AGENT のビット列と OBSTACLE のビット列 4 を＆演算で結び、更に＆＆演算で結ぶ。こちらはビットの and ではなく条件の and なので、&ではなく&&を使う。&演算の結果によって、以下のような処理を行う。

- model.data.get(cood)&GridWorldModel1.AGENT != 0
  → AGENT がいる場合に青い円を描くメソッドを呼ぶ

- model.data.get(cood)&GridWorldModel1.OBSTACLE != 0
  → OBSTACLE がいる場合に黄色に塗るメソッドを呼ぶ
- (model.data.get(cood)&GridWorldModel1.AGENT != 0) &&
  (model.data.get(cood)&GridWorldModel1.OBSTACLE != 0)
  → AGENT と OBSTACLE がいる場合に黒に塗るメソッドを呼ぶ

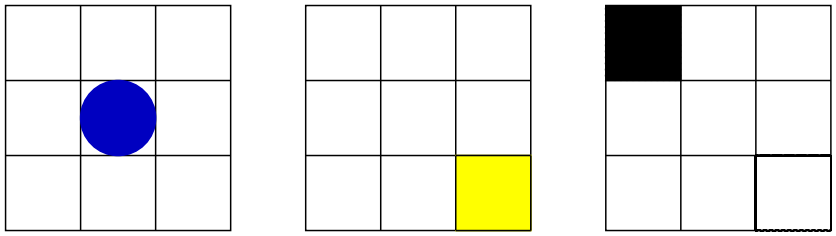

図 3: オブジェクトの描画

この結果、図 3 右のように描画される。このように、HashMap を有効に利用することで、設定した全ての状態を描画することが可能になり、同じマス目に複数のオブジェクトを重ねて描画できない問題も解決された。

3.3 Map の動的拡大

Map 内のマス目を描画するにあたり、Map 全体の大きさを 700 ドット× 700 ドットで描画するように設定した。1 マスごとの縦横のサイズは、以下のような式で計算した。

- 1 マスあたりの横サイズ = 全体の横サイズ / 横方向のマス目数
- 1 マスあたりの縦サイズ = 全体の縦サイズ / 縦方向のマス目数

当初、Grid の左上を $(0, 0)$ と設定し、Map が拡張されれば、x, y 座標の最大値を拡張方向に応じてそれぞれを更新するという形を取った。しかしこれでは座標の最小値が 0 に固定されていたため、Agent が負の方向に移動する際、Grid が描画されない問題が生じた。そこで新たに、4 変数を用いて Map 上下左右の端を表現した。初期設定として最小値は $(0, 0)$ に設定していたが、Agent が負の方向に動く事によって、徐々に x, y 座標の最小値をそれぞれ更新されていくようにした。例えば、初期設定で Agent が 2 体・マス目の数を横 3 マス縦 3 マスとすると (この時点では左上の座標は $(0, 0)$)、Agent が y 座標の負の方向に 1 マス動いた時、$3 - (-1) = 4$ となり横 3 マス縦 4 マスの Map が描画される。このように変更することで、HashMap を使用して負の座標を表現することが可能になった。(図 4, 図 5 参照)

また (図 6) のように、新しく拡張された範囲の描画は、HashMap クラスのメソッド containsKey を使用した。containsKey とは、要素を指定するためのキーが Map 上に存在するかどうかを確認することができ、存在していたら真を返す機能である。Map が拡

張された場合、Agent がまだ通っていない座標は、HashMap のキーとしてはまだ存在していない状態である。そこで、containsKey メソッドを使って、Map の範囲内にあるが、containsKey でその位置のキーがないと判定された位置に対しては、まだ開拓されていないことを表す赤いエリアが描画されるようにした。

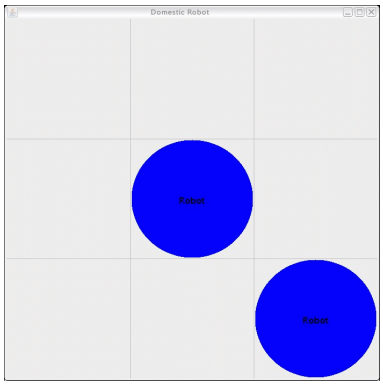

図 4: 3 × 3 マスの描画

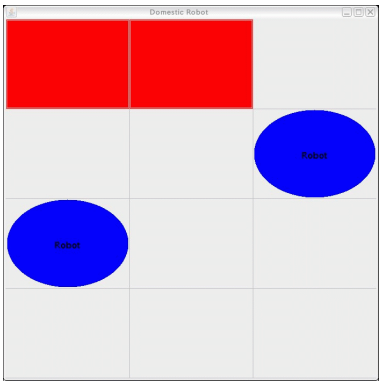

図 5: 3 × 4 マスの描画

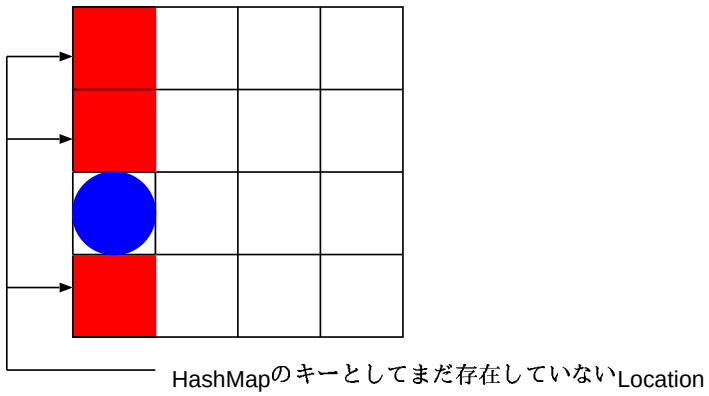

図 6: containsKey を使用した描画

4 検証実験

本節では、前節までに述べた我々が実装したシミュレータを利用した例題について説明する。検証実験として、マップ内に存在する Treasure を発見する事を目標とする例題を、我々のシミュレータを用いて作成した。ただし Tresure は未開拓領域中にランダムに発生し、エージェントが Treasure のマスに移動すると Treasure を獲得したことになり、Treasure は黒に描画される。未開拓領域は赤に描画されるが、エージェントが未開拓領域のマス目に移動し、何もないと知覚した時、白いマスが描画される。

図 7: ジェネレータの GUI

図 8: GUI に初期値を代入した後の様子

[6] で述べるジェネレータによって、(図 7) の画面が表示され、初期値としてマップの横幅を表す WIDTH と、マップの縦幅を表す HEIGHT の値、エージェントの数を入力できるようになっている。そしてそれぞれの Agent の配置を決定すると (図 8)、設定した場所 (この場合 2 体の Agent が $(1,1)$ と $(2,2)$ に配置されている) に Agent が描画される (図 9)。描画された Map 内の端まで Agent が進むと、新たに拡張された Grid が描画され Agent は未開拓な場所へ進むことが可能となる。この時、一度も通っていないマス目は赤色、Treasure は黄色、一度通った Treasure は黒色に描画している。検証実験により、シミュレーションを実行し、観察を行った結果、Agent が負の方向に進んでも描画の問題は特になく (図 10)、再描画も問題なく進行し、徐々に Map は拡張されることが確認できた。(図 11)

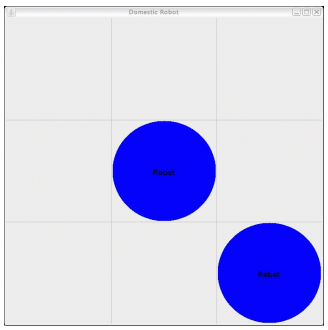

図 9: 初期設定の描画

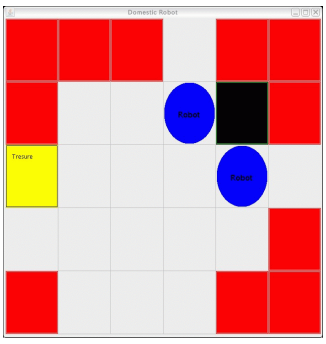

図 10: 例題のシミュレーションの様子 1

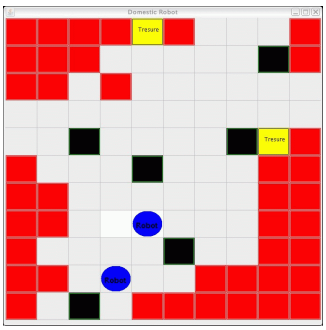

図 11: 例題のシミュレーションの様子 2

5　まとめ

我々は Jason において、Map の大きさが固定されない実世界に近い設定をシミュレーション環境で実現でき、しかも様々な初期設定でシミュレーションが行えるようなシミュレーション環境の開発を行った。４章での検証実験により、textfield に入力した Map の横幅 WEIGHT と縦幅 HEIGHT の幅・Agent の数・初期位置が正しく描画され、同じマス目にオブジェクトが重なっても正常に描画されることが確認された。また Agent が負の方向に進んだ時も、Map の再描画・Map の動的拡大は問題なく描画された。

今回使用した HashMap は、Map を正負の方向に拡張しても、位置を表すキーと場所を表すビット列データを効率よく対応づけできるので、実世界空間に近付けた表現をすることが可能である。また、メモリの制限以外に Map の大きさの制約はないため自由度も高い。

将来の展望として、Map 拡張機能の終点の決定 (今回のシミュレーションを実世界上においてある目標に向かって動かした時、全く無関係な方向へ進まないように規制するという意味での終点)、そして複数の Agent がいる場合において、各々の Agent に信念を持たせることによって、より複雑な行動を可能とすることなどが考えられる。

謝辞

本論文を作成するにあたり、指導教官の新出 尚之准教授から、丁寧かつ熱心なご指導を賜りました。そして藤田 恵先輩、片山 寛子先輩やゼミの同期からもたくさんの助言を頂きました。強力していただいた皆様へ心から感謝の気持ちと御礼を申し上げたく、謝辞にかえさせていただきます。

参考文献

[1] Rafael H. Bordini, Jomi Fred Hübner and Michael Wooldridge: *Programming Multi-Agent Systems in AgentSpeak using Jason*, John Wiley & Sons (2007).

[2] http://java.sun.com/javase/ja/6/docs/ja/api/java/util/HashMap.html

[3] http://www.gpgsim.net/gpgsim/comp-mas.html「マルチエージェントシミュレータ比較」

[4] http://www.soars.jp/ja/index.html「SOARS Project」

[5] http://platbox.sfc.keio.ac.jp/「PlatBox」

[6] 岡田英里「マルチエージェントシステムの Jason シミュレーターにおける汎用的拡張ジェネレータ作成」奈良女子大学理学部情報科学科 2010 年度卒業論文，2011.