マルチエージェントシステムの Jasonシミュレータ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 07251603 岡田英里

平成 23 年 2 月 10 日

概 要

マルチエージェントシステムにおいて、システム全体の振る舞いは、エージェント同士が相互に作用することによって決定されるので、複雑で予測するのは難しく、開発は困難である。そこで、マルチエージェントシステムの開発においてはシミュレーション環境が充実していると有利である。

本研究では、エージェント開発用フリーソフトウェア Jason のサンプルプログラムである domesticRobot を基に汎用化することによって、特にマルチエージェントシステムに関する問題設定としてよく現れるグリッドワールドについて、そのシミュレーション環境を構築するジェネレータを開発した。これにより、グリッドワールドにおけるシミュレーション環境が容易な操作で生成できるようになった。

本論文では既存のサンプルプログラムの問題点について述べ、さらに本研究で構築したシミュレーション環境を構築するジェネレータの開発について述べる。

1 はじめに

マルチエージェントシステムとは、多数の自律的に行動するエージェントから構成されるシステムである。個々のエージェントでは困難な課題をシステム全体として達成する。それぞれのエージェントは自分の環境を知覚し、自分の目標を達成するように行動をとる。エージェントを集中的に管理するものは存在しない。システム全体の振る舞いは、エージェント同士が相互に作用することによって決定される。また、この振る舞いは各エージェントの行動決定に影響を及ぼす。エージェントたちはこのフィードバックから、自分の行動を変化させなければならない。それぞれ異なった判定アルゴリズムなどの特徴を持ったエージェントモデルを用い、複数かつある一定以上のエージェントを多数設定することで、人工社会を構成し、それらの相互作用によって様々な個性を持った人間社会などの予測不可能な事象をモデル化し、できるだけ現実に起こりうる状況をシミュレーションすることが可能である。

しかし、システム全体の振る舞いは、エージェント同士が相互に作用することによって決定されるので、複雑で予測するのは難しい。そこで、マルチエージェントシステムの開発はシミュレーション環境が充実していると有利である。特に、対処すべき問題の設定や規模などを変更しやすいことが望まれる。また、シミュレーション環境が充実していれば、実世界エージェントの開発にも有効と考えられる。

マルチエージェントシステムの開発環境として Jason[1] というものがある。しかし、Jason には上記の条件を満たすようなシミュレーション環境は提供されていない。

本研究では、エージェント開発用フリーソフトウェア Jason のサンプルプログラムを基に汎用化することによって、特にマルチエージェントシステムに関する問題設定としてよく現れるグリッド

ワールドについて、そのシミュレーション環境を構築するジェネレータを開発し、ワールドの大きさの動的拡張を行った。以下、ワールドの大きさを Map と表現する。なお、本研究は本学 4 回生の岸上との共同研究であり、そのうち Map 拡張による実世界空間の表現については [2] で述べる。著者は 3 節に述べる一部分を担当した。

これにより、グリッドワールドにおけるシミュレーション環境が容易な操作で生成できるようになった。新たな問題に対応するシミュレーション環境を作成する際、大きさやエージェント数などを可変にする部分のプログラミングによる記述なしで、それらを GUI で設定できる上、実行時にエリアの大きさも動的に拡張されるので、シミュレーション環境の作成者は、問題の本質的な部分の記述に集中することができる。

2 Jason とは

Jason とは BDI アーキテクチャ[3] を基礎とした AgentSpeak を拡張するためのインタプリタである。Jason は目標を達成するために信念ベースとプランライブラリを用いて意図を生成し、行動を行い、環境と相互作用することによって新しい知覚を得る。これを繰り返し、エージェントは目標を達成しようとする。エージェントを定義するエージェントプログラムは Prolog ライクな文法で書かれ、そのエージェントが置かれる環境モデルは Java を用いて実装する。環境設定プログラムによって環境プログラムを指定することにより、その環境とエージェントの相互作用が可能となる。

2.1 Jason のシミュレーション環境

Jason のシミュレーション環境は Java を用いて実装する。Java を用いるのは、シミュレーション環境で行動する実体を実装するのに、エージェント指向プログラミングによる抽象化がしやすく、また、グラフィカルユーザインタフェースなどの環境モデルに必要なものを提供できるからである。

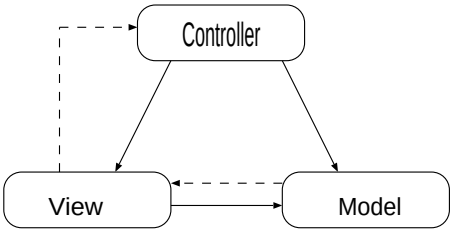
2.2 Json のサンプルプログラム

本研究では Jason のサンプルプログラムとして 1 節に述べた domesticRobot を用いた。domesticRobot は Agent が Fridge からビールを取り、Owner に運ぶシミュレーションである。Owner がビールを欲しいと思ったら、Robot にメッセージを送る。Robot はそのメッセージを達成目標として行動を始める。ビールを Fridge に取りにいき、Fridge を開け、ビールを取り、Owner のところまで行って、ビールを渡すといった行動をとり、Owner からメッセージが送られる限り Robot は動き続ける。

domesticRobot のシミュレーション環境では、Model-View-Controller という技法を用いている。Model-View-Controller とは、コンピュータ内部のデータをユーザに提示し、それに対してユーザが何らかの指示を出すタイプの独自のユーザーインタフェースをもつアプリケーションソフトウェアを、model・view・controller の 3 つの部分に分割して設計・実装するという技法である。model は環境状態の情報や環境の動的変化を維持し、view は model で与えられたことを視覚化し、controller は model と view の情報を伝えあう役割をしている。Model-View-Controller を用いることにより

機能ごとの分離が明確になり、それぞれの独立性が確保される。また、コンポーネント間の依存性が最小限に抑えられるため、他の部分の変更による影響を受けにくい実装にすることができる。

Model-View-Control の概念図は以下のようになる。直線は直接的な関連を表し、破線は間接的な関連を表す。



2.3 サンプルプログラムにおける問題点

domesticRobot は典型的グリッドワールドであり、プログラムと他のグリッドワールドの問題のシミュレーション環境へ転用することも本来容易と考えられる。しかし、プログラム上では初期設定で 10 × 10 のマス目、Agent は 1 体、Owner、Fridge は固定されており、決められた環境内でしか動作することができない。従って、他の問題のシミュレーションに転用するには、プログラムを修正することによってそれらの条件を変更しなくてはならない。シミュレーションを行う際、想定したい環境を自由に変更することができないのは不利である。特に、Agent が実世界の人間をシミュレートしたものだと考えると、人間が決められたマスの中でしか行動しない・できないということはあり得ず、不自然である。

3 シミュレータ設定について

我々のジェネレータは、先に述べた domesticRobot を基にして改造し、GUI による設定変更を可能とする形で実装した。

本章では我々が実現したジェネレータの設計、および実現された機能について述べる。

3.1 プログラム設計

本節では、追加した 4 つの機能について述べる。

- GUI 作成

  GUI を作成するためのプログラムは swSample.java という名前で、Swing を用いて作成した。Swing で作成した GUI は Java プログラム上で描画されるので、より柔軟な設計が可能となる。また、動作環境に影響されることなく一貫したユーザーインターフェイスを提供できる。

  初期設定を行うための GUI を描画しており、マップの初期設定と Agent 数設定を行うためのボタンや値を入力するための TextField を作成している。図 1 がその図である。WIDTH、HEIGHT、エージェントの数に値が入力され、Agent 数決定ボタンが押されたら図 2 のように Agent の場所を入力するための TextField が作成される。

図 1: 値代入前の GUI の様子

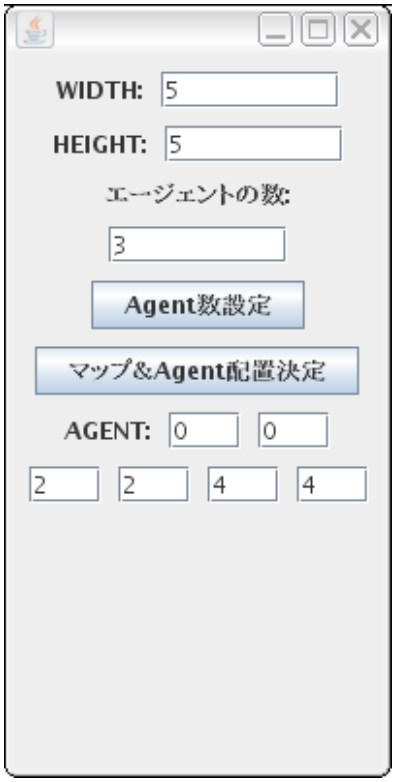


図 2: 値代入後の GUI の様子

マップ&Agent の配置設定は、例えば図 2 のように AGENT に値が入力され、マップ&Agent 配置決定のボタンが押されたら、図 3 のように 5 × 5 のマス目が作られ Agent が入力された位置に配置される。

- GUI に入力された値の受け渡し

  この機能を持つプログラムは configData.java という名前で作成した。swSample の GUI に人間が入力した値を受け取り、Modelutil にその値を渡す。Agent 数は TextField に値が代入されるまで大きさがわからないので、入力された値は ArrayList に格納している。

- GUI によるデータ変更を画面に反映

  この機能を持つプログラムは Envutil.java という名前で作成した。他の全てのプロセスを起動する役目を持つ init と知覚メソッドからできており、model, view, updatepercept に依存している。

  init では config 及び swSample のインスタンスを生成する。そして、サブプロセスの終了を待つ関数を使い、swSample の GUI に値が代入されるまで待機する。次に SampleHouseModel のインスタンスを生成し、GUI を用いて View を作成し、SampleHouseView に SampleHouse-Model のモデルを反映させている。知覚メソッドでは、最初の描画を行っている。

- Agent の場所指定

  この機能を持つプログラムは Modelutil.java という名前で作成した。configData から swSample の GUI に人間によって入力された値を受け渡され、その値から Agent の場所指定を行っ

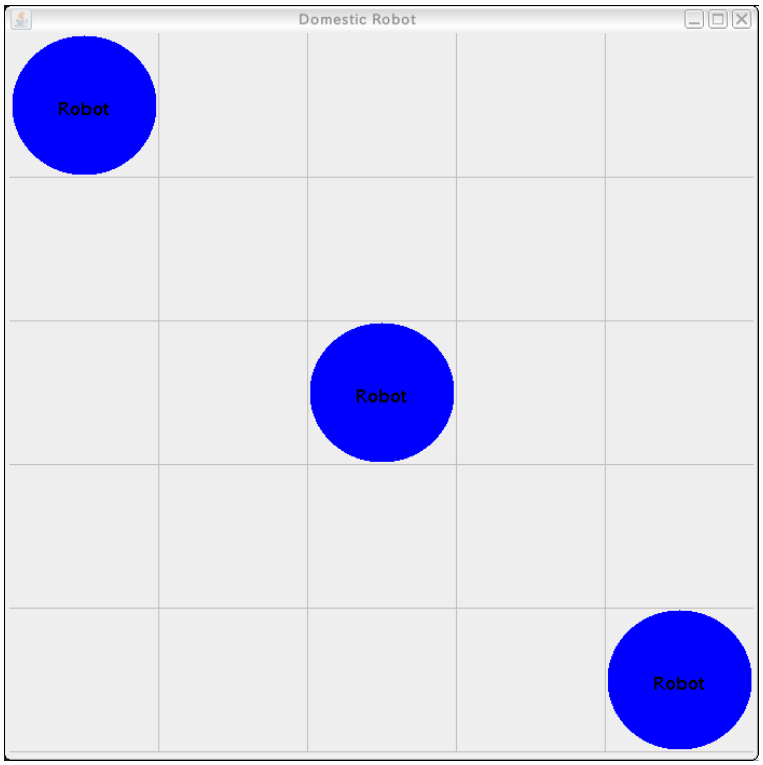


図 3: 生成されたシミュレーション環境

ている。

3.2 クラス設計

今回ジェネレータのために作成したプログラムのクラス設計を示す。UML 図は図 4 のようになっている。以下、簡単に各クラスの説明を述べる。

- GridWorldView1, SampleHouseView： Agent や Obstacle などの描画・視覚化
- Environment, SampleHouseEnv：環境の設定
- Envutil：GUI を用いて view と model の一致
- waitJFrame：window の dispose を行い、値が返ってくるまで待機
- swSample：GUI の描画
- configData：swSample の GUI に人間が入力した値を受け取り Modelutil へ受渡す
- GridWorldModel1, SampleHouseModel：環境状況の情報や環境の動的変化の維持
- Modelutil：Agent の場所指定

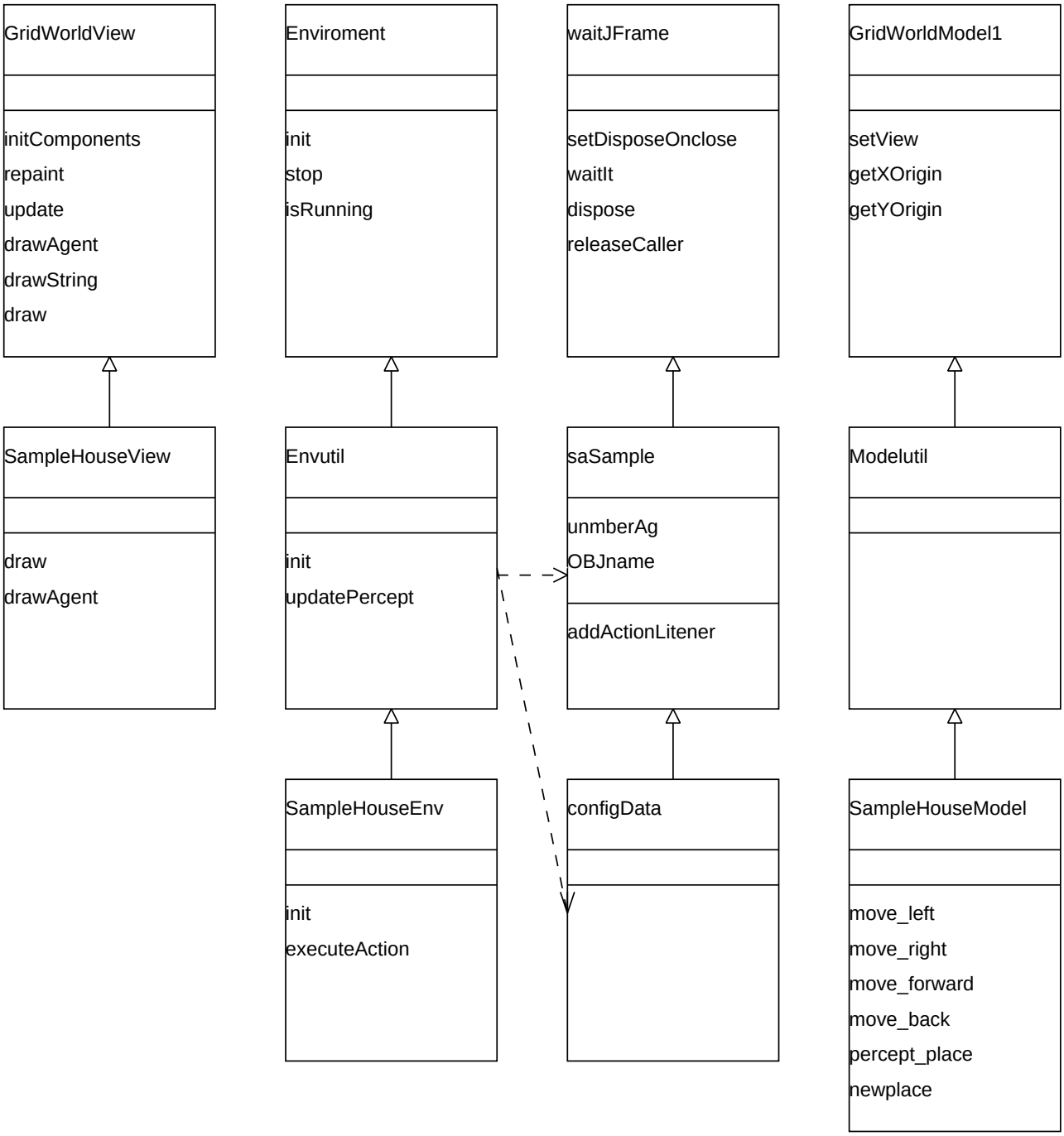

GridWorldView
initComponents
repaint
update
drawAgent
drawString
draw
Enviroment
init
stop
isRunning
waitJFrame
setDisposeOnclose
waitIt
dispose
releaseCaller
GridWorldModel1
setView
getXOrigin
getYOrigin
SampleHouseView
draw
drawAgent
Envutil
init
updatePercept
saSample
unmberAg
OBJname
addActionLitener
Modelutil
SampleHouseEnv
init
executeAction
configData
SampleHouseModel
move_left
move_right
move_forward
move_back
percept_place
newplace

図 4: UML 図

3.3 処理の流れ

Envutil で swSample と configData のインスタンスを生成し、configData に入っている値を swSample に渡す。configData は swSample が用いている TextField に値が代入されるまでは処理を行うことができないので、値が代入されるまで待機する。そして、SampleHouseModel のインスタンスを生成し、GUI を用いて View を作成し、SampleHouseView に SampleHouseModel のモデルを反映させる流れになっている。

4 例題を用いた実験

我々が開発したジェネレータの有用性を示すために以下のような例を作成した。

この例題は、Map 内を動く複数の Agent がランダムに出現する Treasure を発見していくというものである。

以下、この例題について述べる。

4.1 設定

この例題では Agent が Treasure を発見していくものとなっている。初期設定は図 2 に示したようなユーザインタフェースで行う。

マス目の大きさと Agent の個数およびその Agent の配置を設定する。それぞれの値を代入してボタンを押すと、Agent は動き出し、自由に Map を拡張していきながら動きまわり、Treasure を探す。新しく拡張された土地は赤く塗りつぶし、Treasure は新しくできた Map 上にも出現するように設定している。Agent が Treasure と重なったとき、Agent が Treasure を発見したとみなし、そのマス目を黒く塗りつぶしている。Agent と重なっていない Treasure はピンク色に塗りつぶしている。

4.2 結果

図 5 にジェネレータによって生成された環境における実際のシミュレーションの様子を示す。図 5 は Agent が 3 体いる場合を示している。

このようにユーザが望む規模のシミュレーションを行う環境が容易に生成でき、より実世界に近いシミュレーション環境が実現できている。

この例題はあくまで検証用のものなので、問題の設定は単純であり、以下に示すような問題点がある。1 つ目は、まずゴールが決められていないので終わりがない。Agent が Treasure を探し続けるだけとなっており、Agent が Treasure を発見する個数に制限はない。さらに、発見個数をカウントしていないので永久に続くゲームとなっている。ジェネレータに Treasure 発見個数を入力できるよう変更し、ゴールを設定する必要がある。

次に、Agent が既発見の Treasure と未発見の Treasure を区別していない。Agent と Treasure が重なったとき、重なったマス目が黒に塗られることによって Treasure が既発見であることを示す。しかし、Agent はその情報を見ていないので Agent が既に発見された Treasure である黒に塗られたマス目を何回も通ってしまう。これらを改善することで、より実践的な問題によって、我々のジェネレータの活用性をさらに検証することが必要と考えられる。

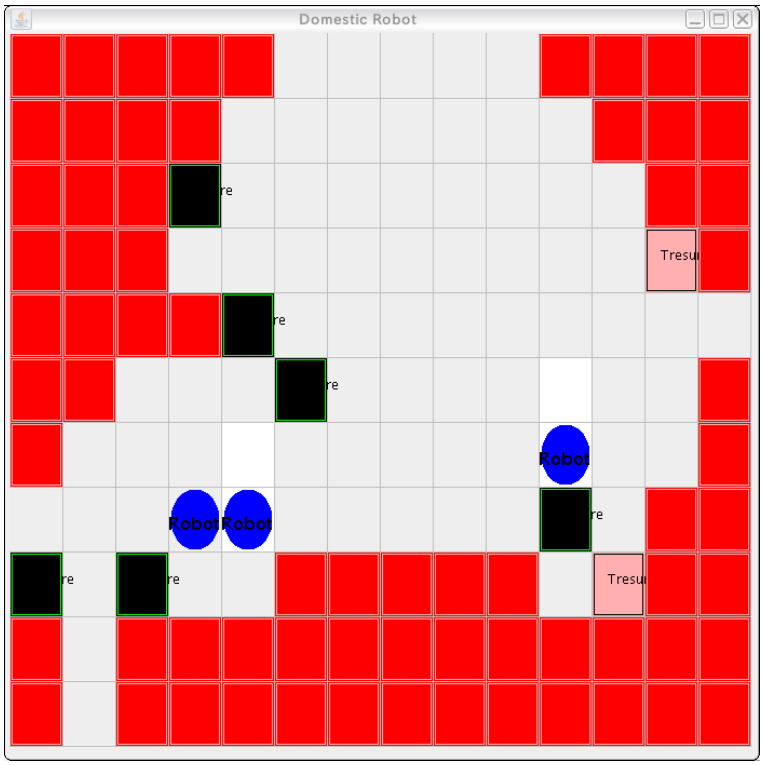


図 5: 例題のシミュレーションの様子

5　まとめ

ジェネレータ作成により、グリッドワールドにおけるシミュレーション環境において、対処すべき問題の設定や規模の設定を容易に設定することが可能になった。それらの設定が、新たな問題に対応するシミュレーション環境を作成する際、可変にする部分のプログラミングによる記述なしで、GUI で行える。また、実行時にグリッドワールドのエリアの大きさも動的に拡張されることからより実世界に近いシミュレーション環境が実現できている。これらにより、シミュレーション環境の作成者は、問題の本質的な部分の記述に集中することができるようになった。

しかし、シミュレーションを行うたびにシミュレーションしたい環境設定を毎回 GUI に手作業で設定して行わなければならないので、様々なシミュレーションを行いたいときに手間がかかってしまう。

将来の展望としては、クリックやドラッグで Agent の配置が可能となり、ファイルから読み込んでシミュレータの環境設定を行うことが可能となれば時間も短縮でき、簡単にシミュレーションができるだろう。

謝辞

本論文の執筆及び研究にあたり、いつも親身に御指導をしていただいた新出 尚之准教授に深く感謝し、厚く御礼申しあげます。また、たくさんの助言をいただいた藤田 恵先輩に感謝の意を表します。

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] 岸上 文,「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」, 奈良女子大学理学部情報科学科 2010 年度卒業論文 (2011).

[3] Munindar P. Singh, Anand S.Rao, and Michael P. Georgeff, : *Formal method in DAI*, Multiagent systems:a Modern Approach to Distributed Artificial Intelligence Research (1999).