実世界におけるエージェント構築に向けた シミュレーション環境の作成

奈良女子大学 理学部 情報科学科 4 回
新出研究室 久妻さゆり

2013 年 2 月 13 日

概要

始めから実世界での実験は困難である。そのため、従来、エージェント及びロボットに関する研究において環境は、離散的な状態空間を用いた仮想世界上のものとして取り扱ってきた。しかし、行為が実世界にて成功するためには、離散的なシミュレータでは不十分であり、連続した状態空間上で、環境の多様な変化に応じて合理かつ柔軟に対応したシミュレータが必要である。

本研究では、実世界においてエージェントを扱う手段の構築を目指すため、カヌーレーシングをテストベッドとし、連続世界に基づくシミュレーションを作成した。現在、シミュレーション上でエージェントは基本行為を行い川を下ることができている。本論文では、シミュレーション環境の設計と実現について述べる。

1 はじめに

人はそれぞれに自分の信念を持ち、状況の変化に動的に対応し、意思決定をすることで目標達成に向けた一貫的な行動をとることができる。この考え方を応用し、信念 (Belief)、願望 (Desire)、意図 (Intention) という 3 つの心的状態パラメータを用いて、合理かつ自律的なエージェントを実現したものが BDI エージェントである。BDI エージェントは人間の思考をモデル化し、エージェントとして実現することで、作業を行う際は常に熟考しながら自律的に行動選択を行う。この方法を用いることにより、エージェントは矛盾のない一貫した行動をとることが可能となる。

そして我々は、自律型エージェントの実装を目指すため、BDI エージェントを用いた研究を行ってきた。従来、BDI エージェントを用いた研究において環境は、グリッドワールドである離散的な状態空間を用いた仮想世界上のものとして取り扱ってきた。そのため、エージェントは決められたマス目を動くといった制限された行動に限られていた。

しかし、実世界にて行為が成功するには、エージェントが決められたマス目でしか行動しない・できないということはあり得ず、不自然である。自由に行動できるためには、連続した状態空間上で、環境の多様な変化に応じて合理かつ柔軟に対応する必要がある。

したがって、今回、実世界への足がかりとして、BDI エージェントの実装手段である Jason[1] を用いた連続世界のシミュレータ作成を目指した。本研究では、カヌーレーシングをテストベッドとし、実世界におけるエージェント構築に向けたシミュレーション環境の作成を行った。なお本研究は、奈良女子大学理学部情報科学科 4 回生濱田との共同研究である。学習と意思決定については [3] で述べる。本論文では、カヌーシミュレーション環境の設計と実現について述べる。

2 カヌーシミュレーション概要

本研究において、作成したカヌーシミュレーションの概要について述べる。

エージェントが川のある地点に置かれるとき、エージェントが基本行為を用いてカヌーを操り、岸に衝突しないように川を下ることを目指す。このとき、エージェントは、自身の位置、進む方向、速さの情報を持つこととする。また、連続空間上で川の流れをそのまま表現すると無限の情報を持たねばならないので、簡略化するため、川を一定区間ごとに区切り、それぞれに水流の向き、速さの情報を持つこととした。エージェントが川を下るまで実行し続ける。エージェントが岸に衝突した場合、エージェントは同じ y 座標上の近くの岸へ移動し、再度その地点から出発する。

3 開発環境

BDI モデルを用いたため、エージェントの記述には Jason[1] を、環境の設定には、Java を用いた。

3.1 Jason

Jason とは、BDI アーチテクチャを基礎として 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せしようとする。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になる。

3.2 Jason のサンプルプログラム

本研究では、カヌーシミュレーション環境の作成において Jason のサンプルプログラムである domesticRobot を参考にした。

domesticRobot とは、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 がビールを飲み、なくなると Robot にビールを取ってくるよう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 に取に行くため、Fridge へ移動し、Fridge を開け、ビールを取り出し、Owner のところへ運び、ビールを渡す。また、Fridge にビールがないとき、SuperMarket へビールを注文するメッセージを送る行動をとる。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は、Env、Model、View の 3 つのクラスから成り立っている。Env はエージェントから plan を受け取り、plan の実行結果をエージェントに返すクラス、Model は Env から渡された plan を実行するクラス、View は Model で与えられたことを視覚化するクラスである。このように記述することで、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ようになる。そのため、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。

そこで、本研究では、Env、Model、View、の 3 つのクラスに基づいて環境を作成した。

3.3 Jason のサンプルプログラムにおける変更点

本研究で作成したカヌーシミュレーションで、そのまま domesticRobot の環境を使うことはできないので、以下の点を変更した。

- グリッドワールドから、実数値に対応できるように変更
  domesticRobot は、離散的なグリッドワールドで記述されているため、連続世界に基づいて記述するために実数値に対応できるように変更した。

- エージェントが自由に移動できるように変更
  domesticRobot では、エージェントは決められた一定区間を行き来することしかできない。しかし、カヌーシミュレーションではエージェントが自由に動く必要がある。そのため、エージェントが座標などの情報を持つクラスを作り、可能にした。

- シミュレーション環境を自由に変更できるように変更
  domesticRobot でシミュレーション環境は、１０×１０マス、Agent は１体、Owner、Fridge といった Agent が目標を達成するために必要な場所、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。そのため、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便である。そのため、川の形状や流速など、環境を定義しているクラスを作り、そこを入れ替えることで環境を変更可能にした。

- エージェントをオブジェンクトにした
  domesricRobot ではエージェントをオブジェクトとして表現せず、各マス目にエージェントや Fridge など何があるのかを整数のビットで表現していた。しかしカヌーシミュレーションでは、実数座標であるため、その実装方法はとることができない。また、実装言語がオブジェクト指向言語である Java であり、エージェントは主体的に動く存在であるためオブジェクトとして実装することが設計上も望ましい。そのため、RiverAgent という、エージェントの座標や速度など、エージェントの情報を持つクラスを作った。

4 プログラム構造

プログラム構造は、図 1 のようになった。本論文では、環境を記述している Java での、設計と実現について述べる。

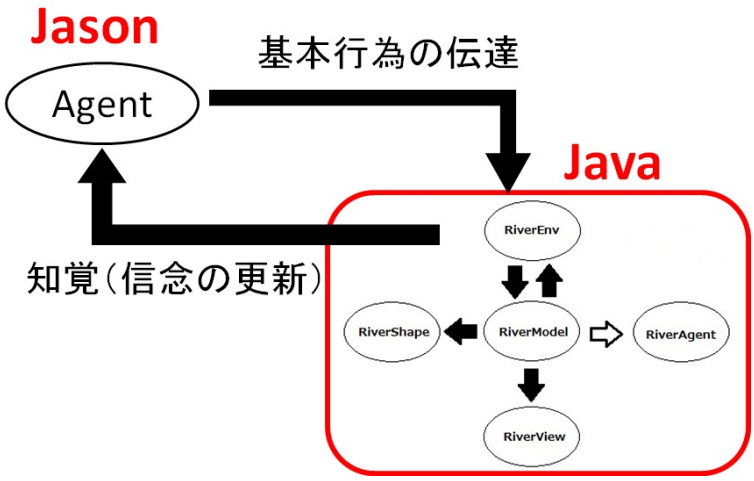

図 1　プログラムの構造

4.1　クラス設計

シミュレータのクラス設計は、図 2 のようになった。図の黒矢印は、情報の伝達と作業の依頼の関係を示し、白矢印は一方が他方を持つ関係を示す。

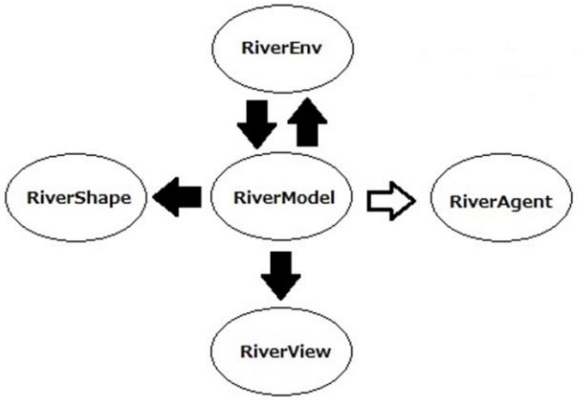

図 2　プログラムの構造

4.2　各クラス概要

以下に各クラスの概要について述べる。

- RiverEnv
  これは、Jason が提供する Environment クラスのサブクラスである。そのため、環境を設定するメソッドであり、Jason とやりとりをするため、エージェントの知覚と基本行為の実行に関するデータやメソッドを持つ。無条件で最初に処理をする init メソッドを持ち、開始時に RiverModel をインスタンス化する。
  また、基本行為の実行は Environment の executeAction で行われており RiverEnv は

executeAction をオーバーライドすることで自前の基本行為を実現している。そのため、基本行為を実行する流れは次のようになる。Jason から基本行為を受け取ると、executeAction で、エージェントは川の流速にそって流された後、RiverModel へ基本行為の伝達を行い、RiverModel から実行結果を受け取る。受け取った結果、エージェントが川の中にいるかどうかを判定し、エージェントへ知覚の更新を行う。

- RiverModel
  これは、RiverEnv から受け取った基本行為を処理するためのクラスである。また、実行時の画面のサイズや、エージェントの最初の位置等の初期設定値を持っている。基本行為を受け取ると、必要な処理をするため、RiverAgent からエージェントの位置情報を取得し、RiverShape へ受け取った位置情報の流速を取得依頼し、エージェントが次に進む地点を計算する。そして、新しいエージェントの位置情報を RiverAgent へ更新し、RiverModel は、RiverView をインスタンスとして持つため、RiverView へエージェントの描画依頼をする。最後に、基本行為の実行結果を RiverEnv へ返す。

- RiverView
  これは、JFram のサブクラスであり、川やエージェントなど描画を担当するクラスである。エージェントの情報が更新される度、RiverModel から、描画依頼を受ける。
  エージェントは図 3 のように、三角形で表し、エージェントの向きにより傾くようにした。更に、赤い印をつけることでエージェントがどちらを向いているかわかるようにした。また、川の流速は目には見えないため、青い線を書くことで表現した。青い線の傾きは、流速の向きを表しており、長さが長いほど流れが速いことを示している。

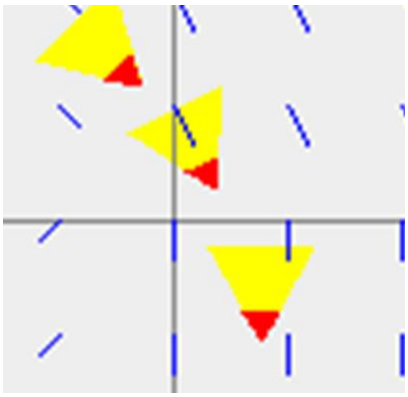

図 3　エージェント描写

- RiverShape
  これは、川の形状や、川の各地点の位置情報、水流の情報を持つクラスである。川の形状を返すメソッド、エージェントが川の中にいるか判定するメソッド、与えられた座標の川の流れを返すメソッドがある。

- RiverAgent
  これは、エージェントの位置情報、進む向き、速さを持つクラスである。

4.3 基本行為の実装

本項では、作成した 5 つの基本行為の実装について述べる。図 4~ 図 8 の青矢印は川の流れを、緑矢印はエージェントがオールを漕いだ力を、赤矢印は合力を表す。

- no_doing(動作の停止)
  これは、エージェントが何も動作をしていないという行為を表す。そのため、エージェントは川に身を任せたように、川の流れがそのままエージェントの新しい地点へ反映される。図 4 のようになる。
- move_toward（前進）
  これは、エージェントが川の流れに沿ってオールを漕ぐ行為を表す。そのため、エージェントは、流速と自身の速度との合力だけ進む。図 5 のようになる。
- move_back（後進）
  これは、エージェントが川の流れに逆らってオールを漕ぐ行為を表す。そのため、エージェントは流速と、それに逆らって漕いだ力との合力だけ進む。図 6 のようになる。
- right（左へ曲がる）
  これは、エージェントが右手だけオールを漕ぐ行為を表す。そのため、カヌーは左へ曲がるため、エージェントは、流速と漕い力との合力だけ進む。図 7 のようになる。
- left（右へ曲がる）
  これは、エージェントが左手だけオールを漕ぐ行為を表す。そのため、カヌーは右へ曲がるため、エージェントは、流速と漕い力との合力だけ進む。図 8 のようになる。

図 4　do_nothing(動作の停止)

図 5　move_toward(前進)

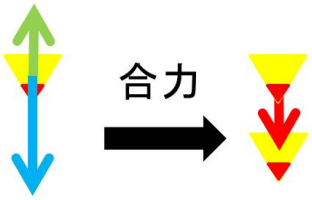

図 6　move_back(後進)

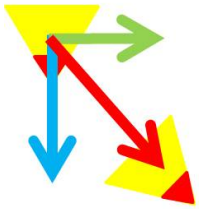

図 7　right(左へ曲がる)

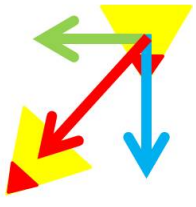

図 8　left(右へ曲がる)

5　実行結果

図 9 ～ 図 13 は、それぞれの基本行為をし続けたときのシミュレータの結果である。全て、同じ環境、同じ地点から、エージェントは川を下った。エージェントが基本行為を行う度に描画した。

図 9 では、エージェントは何も動作しないので、川に身を任せて下る。図 10 では、川の流れに沿って前進するので、図 9 と比べ、川を速く下る。また、図 11 では、川の流れに逆らうので、図 9 と比べても、川を遅く下る。図 12、図 13 では、左右に曲がり続けるので、岸に衝突しながら進んでいる。

図 14 は、move_toward と right を組み合わせたものである。10 回前進した以降、right で、左へ曲がり続けている。

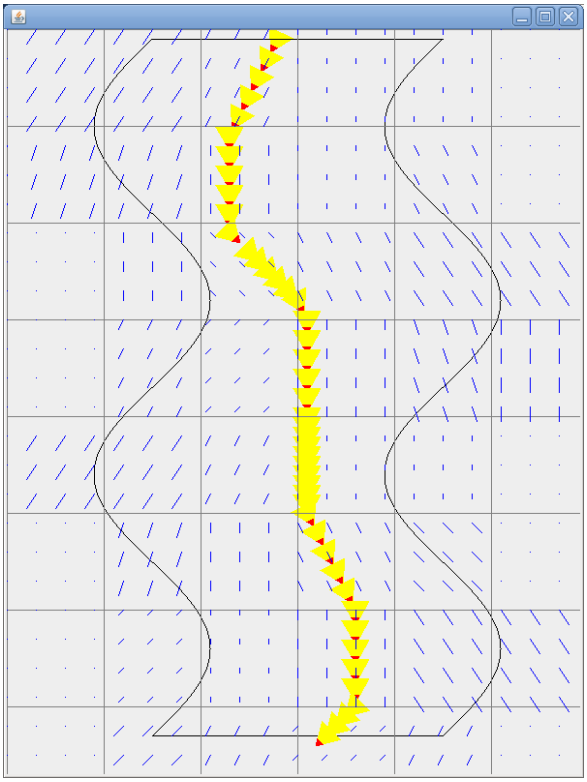

図 9　do_nothing(動作の停止)

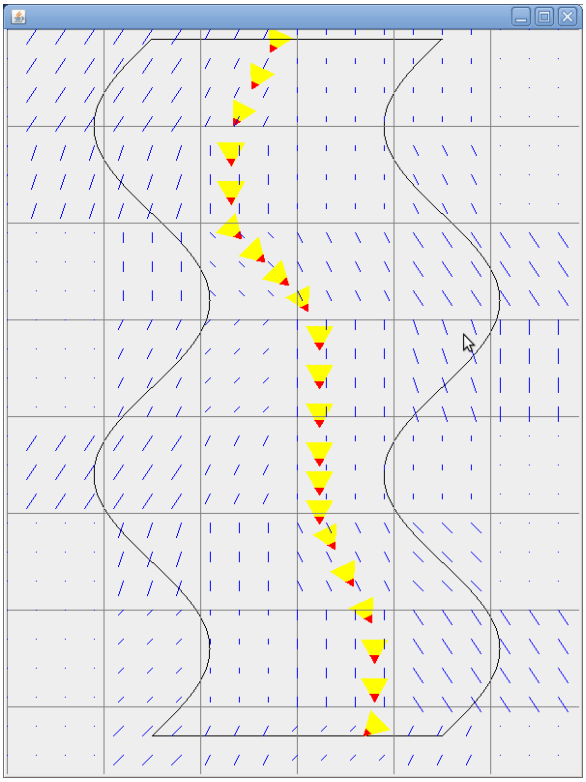

図 10　move_toward(前進)

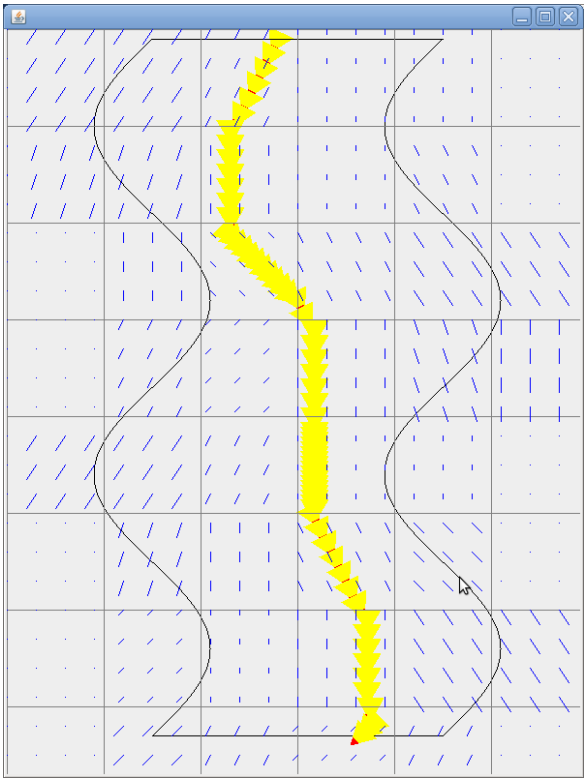

図 11　move_back(後進)

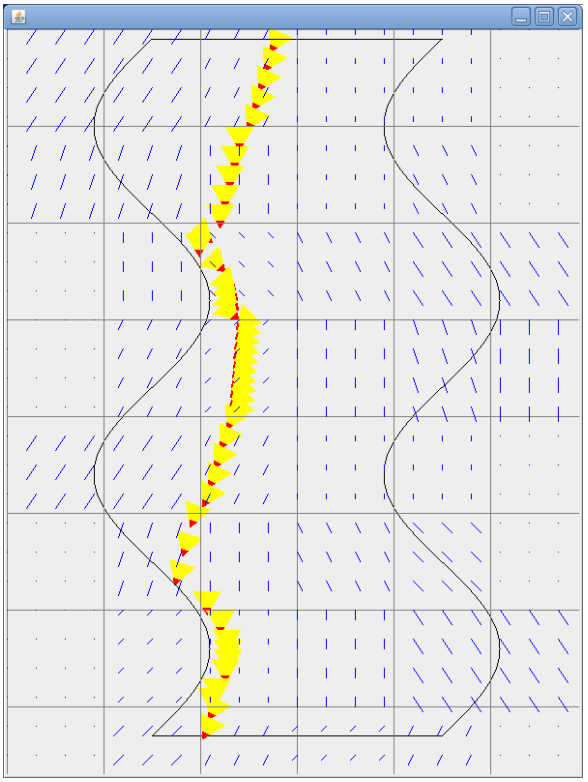

図 12　left(右へ曲がる)

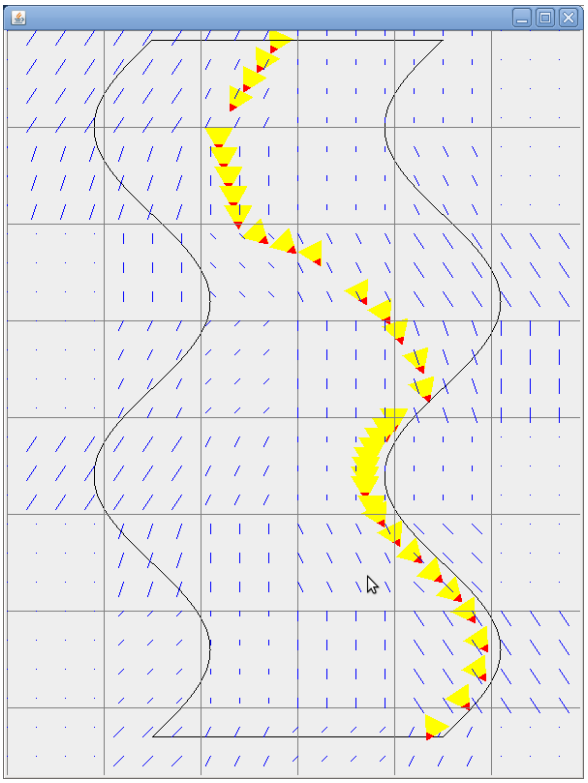

図 13　right(左へ曲がる)

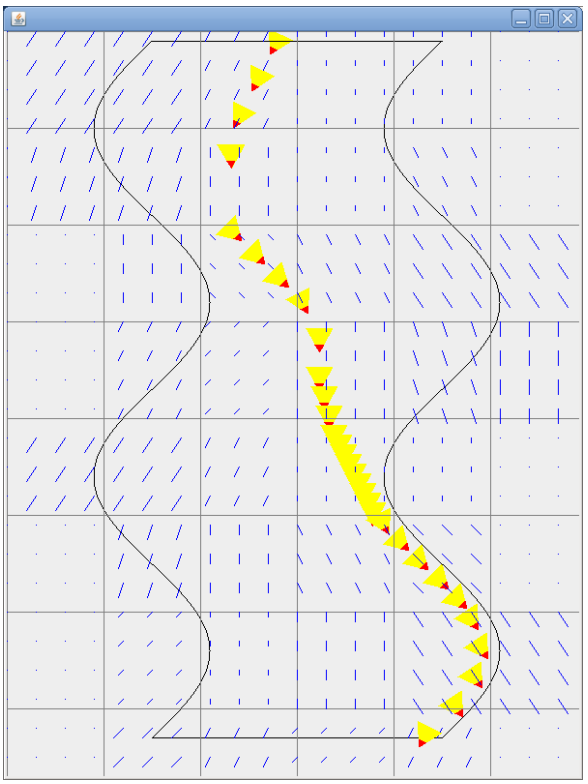

図 14　10 回前進した後、左へ曲がり続ける

6　まとめ

連続世界に基づいたシミュレーション環境を作成することを目指し、カヌーレーシングをテストベッドとし、実世界におけるシミュレーションを作成した。シミュレーションを作成するにあたり、エージェントに必要な基本行為の実装を行い、カヌーシミュレータ上で、基本行為を用いてエージェントは川を下ることができた。

あらかじめプランを記述しておけばそれに従って行為を行うことはできるが、現段階では、目標地点への到達のプランを作ってその目標へ向けて漕ぐことはまだ実装できていない。学習で漕ぎ方を獲得することも今後必要である。

今後の課題として、より実世界に近づけるために、川に岩をランダムに設置するようにし、エージェントが岸や岩に衝突することなく川を下るように学習させる必要がある。そのため、環境の設計と実現の観点から、以下の点を考察する必要がある。

- シミュレーション環境 (川) の時間変化を実装する
  本研究では、連続空間である川の環境を簡略化するため、格子状に区切ったが、実際はそのようになっていない。時間変化に伴い水流も変化する。より実世界に近づけるためには実装すべきことである。
- 行為の開始時から終了時までに発生するタイムラグ
  本研究のシミュレータでは、川の流れと基本行為の実行は別々に行われ、時間差がないように実装されている。そのため、エージェントが右へ曲がるとき、右向きの力と水流との合力で斜めに進むようにした。しかし実際には、開始時から終了時まで、川に流されながら行為を実行するため、タイムラグが生じる。そのため、エージェントは斜めに進むのではなく、弧を描くように進むはずである。
- 情報取得のためのセンサ
  エージェントは物体を目視することができない。そのため、衝突を回避するために、環境の情報を取得する何らかのセンサが必要である。しかし現実のセンサには、1. 情報獲得に時間がかかる、 2. 得られる情報が完全でない、 3. 物理デバイスは外乱や故障が起こる可能性がある、 4. 複数のタスクを並行して扱わなければならない、など様々な問題がある。このような現実に近づけた実装をする必要がある。[2]
- 川の流れやカヌーの漕ぎ方などをより現実に近づけたモデルにする
- エージェントを任意の点から出発させる
- 障害物に衝突した後の処理
- 川の中にランダムに岩を配置する

以上のことをシミュレータに反映させる必要がある。

7　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していただいた指導教官の新出尚之准教授に深く感謝し、厚く御礼申し上げます。また、新出研究室のみなさまに、感謝の意を表します。ありがとうございました。

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] 高田司郎, 新出尚之. 行為のアトラクター状態を考慮した知能ロボットについて. In Proc. of JAWS2012, 2012.

[3] 濱田 百合. 連続世界におけるエージェントの学習と意思決定に向けて. 奈良女子大学理学部情報科学科 2012 年度卒業論文 (2013).