感情表現を用いて行動決定を行うエージェントの実現

奈良女子大学 理学部情報科学科 4 年
新出研究室 清水詩子

概要

近年、エージェントへの感情表現の付与が重視されている。本研究では、従来研究において実現されていなかった、感情表現と実世界における行動決定の両方を行うことのできるエージェントの実現を目指して、OCC theory と呼ばれる感情をモデル化した理論における感情を部分的に BDI エージェントに実装し、感情表現を用いた行動決定をさせる実験を行った。本論文では、実装した感情を用いてエージェントが行動決定を行えるようになった実験結果を示し、さらに実装した感情表現の妥当さの検証と、実験結果から判明した今後の問題点について述べる。

1 はじめに

近年、人との対話を実現させる上で、より人間に近いエージェントが求められている。それに伴い、エージェントに感情表現の付与が重視されている。

従来、エージェントに感情を与える試みは広く行われている ([9], [8] 等)。けれども、これらで扱われているエージェントは、人間とのコミュニケーションを行うようなものが主体であり、感情表現を持つエージェントが、なおかつ動的環境下で問題解決を行う能力を持つようなことは従来重視されていない。しかし、人間と同じ空間、つまり実世界で行動するエージェントが人間にとっての親しみやすさを備えることは今後必要になると考えられる。

実世界のような動的環境下で問題解決のために行動するエージェントの実現においては、BDI アーキテクチャが有効であることが示されている [7]。BDI アーキテクチャとは、BDI モデル [6] による行為決定方式を計算機上で実現したものであり、これにより BDI エージェントという人間の合理的思考に基づいた行動をとるエージェントを構築することが可能である。BDI モデルとは、信念 (Belief)、願望 (Desire)、意図 (Intention) の 3 つの心的パラメータを用いた人間の合理的行動、つまり整合的な行動をモデル化したものであり、BDI logic という論理モデルをもつ。

一方で、OCC theory [4] と呼ばれる、心理学的見地に基づく感情のモデル化に関する理論が感情の形式化に多く用いられている。特に、OCC theory は信念や願望といった心的状態を用い、感情の特徴付けが明確である。またその感情の特徴付けを論理モデルで表現可能な形で行える特徴を持つため、BDI logic を持つ BDI モデルとは親和性がよく、特に既存の BDI アーキテクチャの実装である Jason [2] 上での信念ベースを用いて実現が可能である。

そこで本研究では、BDI logic への OCC theory に基づく感情の実装を行った。具体的な実装としては、Jason 上で OCC theory における感情の定義を行うことにより、感情生起の実現と感情を行動決定に用いることができるようにした。

エージェントに感情表現を付与する利点としては、人間とエージェントが対話するにあたり、単なるパターンマッチではなく、人間の感情にエージェントが共感できるようになれば人にとってのある種の癒し、さらに進んで最近その人口の増加が明らかになっているコミュニティ障害等を抱える人々のリハビリになり得ることが期待される。

2 関連研究

BDI モデルと OCC theory を用いてエージェントの感情を扱っている関連研究としては、[1] や [3] がある。

Adam ら [1] は、BDI モデルでの形式化に用いられる論理体系である BDI logic に対し、「不確実だが起こると期待されている事柄」を表現する等いくらかの新たなオペレーターを導入して拡張し、これを用いて、OCC theory で扱われる 22 種類の感情の特徴付けを論理式として形式化することで BDI モデルに取り込んでいる。

ただし、この研究におけるモデルでは一つの行動が一つの感情を生起させるようになっており、複合的な感情の生起については扱えない。また、感情の度合いについても扱っていない。

加えて、現段階の実装としては、状況からの感情の生起を推論する部分を独自に実現したものはあるが、他のシステムと結合できるよう汎用的にまとめられたものはなく、感情表現をエージェントの行動決定の一部として取り込めるようにはなっていないため、BDI エージェント部分の実装を一般利用者が行えない (公開され

ていない)。

そこで、本研究では基本的にこの形式化を踏まえつつ、5 章でのべるように Jason による実装を独立に進めた。これによってエージェントシステムに感情表現を組み込む汎用的な手法として利用出来るようにする。

この他、[3] では OCC theory を用いた BDI エージェントによって、学生の勉学を奨励させる教育用エージェントシステムを実現している。これは、OCC theory を基にした推論システムによって生徒の感情を推論することで、適切な指導を行うことを目的としたものであり、生徒の勉学に対する姿勢を 2 つに大別した上で、それを基にすべての生徒に起こり得る状態を列挙し、感情の流れを記述している。しかし、人間の感情の推論を目指したものであり、エージェントやロボットの感情の生起を扱うものではない。

他には、OCC theory を使わず、独自に感情の概念を取り入れた BDI エージェントを構築したもの [5] もあり、これは特に恐怖の感情にのみ重点的に形式化することを行っている。これは [1] とは違い、いくつかの行動の組み合わせによって感情が生起されるようになっている上、恐怖の感情に対して度合いを設けている。しかし、今のところ恐怖の感情のみを形式化し他の感情にはふれておらず、また現時点では形式化のみで実装には至っていない。

3 OCC theory

OCC theory [4] は、Ortony, Clore, Collins らが提唱した心理学的見地を基にした感情タイプをモデル化した理論で、計算機科学の分野において広く用いられている。計算機上で実装でき、感情の特徴付けが限られているため、比較的理解しやすい。

OCC theory においては、人間の包括的な 22 種類の感情を扱う。それら 22 種の分類をまとめると次のようになる。

1. 事象の結果の望ましさにのみ焦点を当てたもの
   - (a) 結果の望ましさに関して (自分にとって)
     - i. 起こった事象に関するもの
       - ··· Joy(喜び), Distress(悲嘆)
     - ii. 予想に関するもの
       - A. 単なる予想に対して
         - ··· Hope(望み), Fear(恐れ)
       - B. 予想していたことが起こったことに対して
         - ··· Satisfaction(満足), FairConfirmed(恐れていた事が確定)
       - C. 予想していたことが起こらなかったことに対して
         - ··· Relief(安堵), Disappointment(落胆)
   - (b) 結果の望ましさについて (他者にとって)
     - i. 他者が良い結果を得た時
       - ··· HappyFor(共に喜ばしく思う), Resentment(憤りを感じる)
     - ii. 他者が悪い結果を得た時
       - ··· Gloating(ほくそ笑む), SorryFor(共に残念に思う)
2. 行動に対する賞賛度にのみ焦点を当てたもの
   - (a) 自分の行動に関するもの

··· Pride(自尊心), Shame(羞恥心)

a' Joy と Distress との混合型

··· Gratification(満足), Remorse(自責)

(b) 他者の行動に関するもの

··· Admiration(賞賛), Reproach(非難)

b' Joy と Distress との混合型

··· Gratitude(謝意), Anger(怒り)

3. 対象物に対する好き嫌いにのみ焦点を当てたもの

··· Love(好き), Hate(嫌い)

また、これらの感情は常に単独で起きるわけではない。例えば、自分とその友達が同じ職に申し込んだとする。友達の履歴より自分の履歴の方が優れているという信念を自分では持っているが、その後自分は、友人が履歴を多少誇張したことによって自分は不採用で友達が採用となってしまった事実を知る。このとき OCC theory によれば、自分には Disappointment(期待に反して自分が受からなかったため)、友人に対する HappyFor(友人が受かったため)、Reproach(友人の行為に対する非難) の 3 つの感情が生じる。このように、相反する感情でも、原因となるものが異なれば同時に起こり得、そのうちどれが相対的に重要視されるかは、各感情の生起の原因となった事象 (イベントの結果) やアクションの望ましさの度合いによって決まる。

4 BDI エージェント

本研究では、OCC theory を用いたエージェントの感情表現の実装のための基盤として、BDI アーキテクチャを用いる。その理由は、1 つには感情表現は高度に人間らしい性質であるので、人間の行動様式に似せた行為決定を実現する仕組みに組み込むことが適切であると考えられることである。もう 1 つとしては、1 章で述べたようにどちらも論理モデルを利用可能であるため親和性が良いことが挙げられる。特に、OCC theory による感情の特徴付けは、あることを自身が望んでいるかどうか等の心的状態や、その時間的変化を用いて定義されており、それらの多くは、BDI モデルが最初から明示的にもつ、心的状態やその時間的変化という概念によって表現できる。よって、BDI エージェントの実装に用いている手段を、ほぼそのまま感情の実装手段として用いることができる。

本節では BDI エージェントと BDI アーキテクチャ、並びに本研究で BDI エージェントの実装のためのプラットホームとして利用した Jason について述べる。

4.1 BDI エージェントと BDI アーキテクチャとは

BDI エージェントとは B(信念), D(願望), I(意図) という 3 つの心的パラメータを持ち、その時間変化を用いて意思決定を行う自律的エージェントである。BDI エージェントはこのような心的状態を持ち合わせているため、感情のような性質を取り扱うことに有効である。

BDI エージェントの動作は、BDI インタプリタと呼ばれる図 1 のようなループで表される。

初期条件が与えられると、`option-generator` がまず `event-queue` を読み、現在の信念などを用いて、現在実行可能な意図の候補のリスト、`options` を返す。次に `delibetrate`(熟考) を用いて `option` から次の時刻に実行すべき意図を選んで決定し、自らの意図を更新して次に実行すべき行為を決定する (`update-intention`)。決定された行為が基本行為であればそれを実行し (`execute`)、そうでなければそれは新たな副目標であるの

```
BDI-interpreter
initialize();
do
  options := option-generator(event-queue, B, D, I);
  selected-options := deliberate(options, B, D, I);
  update-intentions(selected-options, B, D, I);
  execute(I);
  get-new-external-events();
  drop-successful-attitudes(B, D, I);
  drop-impossible-attitudes(B, D, I);
until quit.
```

図 1　BDI インタプリタ

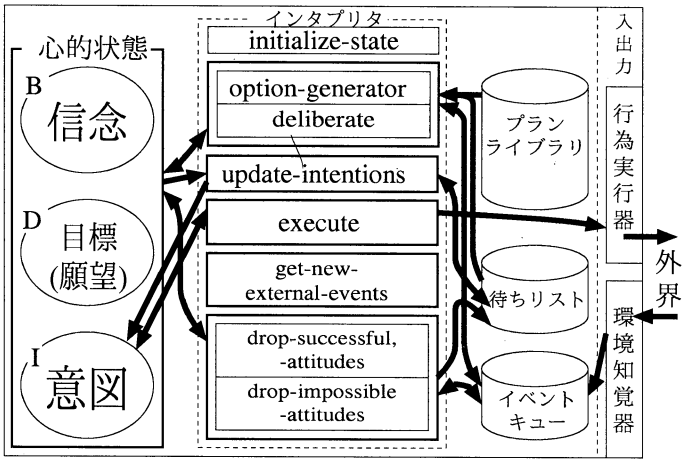

図 2　BDI アーキテクチャ

で、event-queue に追加しておく。続いて環境知覚を行い (get-new-external-events)、成功した、あるいは不可能だと判断された意図を捨てる (drop-successful,impossible-attitudes)。これらの動作を繰り返すことで目標達成に向けた行動を行う。

4.2　Jason とは

本研究では BDI エージェントの実装に Jason を使用した。Jason [2] とは、BDI アーキテクチャに基づくエージェント記述言語 AgentSpeak(の拡張) の処理系であり、4.1 に述べた BDI インタプリタの実現である。エージェントのプランや信念は Prolog に似た文法を持つルールの形で宣言的に記述し、環境モデルやエージェントと環境の相互作用に関する記述 (特にエージェントの環境知覚や基本行為の定義も含まれる) の定義は Java 言語で行う (図 3)。定義された環境知覚や基本行為はプラン内で用いることができる。

プラン記述は基本的に

目標

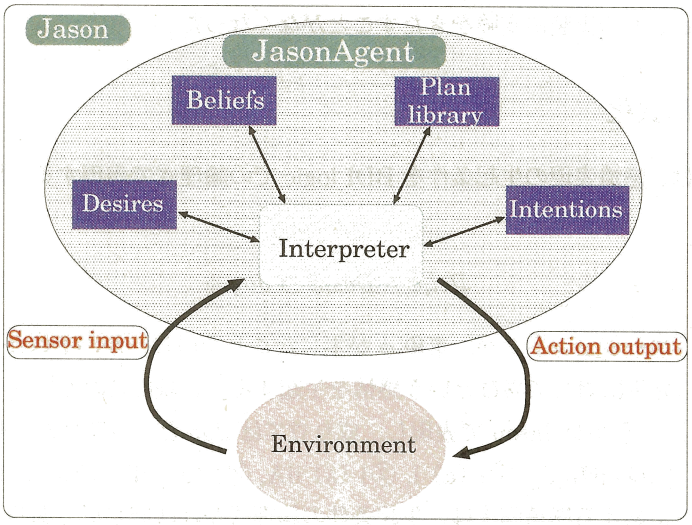

図 3　Jason と環境との相互作用

: プランを適用するための前提条件
<- プラン本体 (基本行為または副目標の列)

という形をとる。プラン頭部の「目標」は実際には、その目標を生じさせるイベントの発生として記述される。Jason インタプリタは、イベントが発生していれば、それに合致する頭部を持つプランの中から 1 つ選んで (「目的–手段推論」)、エージェントの意図の集合 (図 1 の `options`) に追加する。次いで現在の意図のうち 1 つを選んで (「熟考」、同図の `deliberate`)、その本体の行為 1 つ (基本行為または副目標) を実行する。これを繰り返す。

5　実装方法

本節では、3 章で述べた感情表現の実装について述べる。

5.1　annotation 利用の際のバグ

現在の Jason(本論文執筆時点の最新版 1.3.6) には、annotation を付加した項の単一化にバグがあったため、そのバグを取り除いた。バグの具体的な内容は、Jason のエージェント記述ファイル (.asl ファイル) において、annotation 付きのゴールをルールの体部の annotation 付きの項と単一化させると、annotation 部の単一化が正常に行われないというものである。例えば、

```
a[p(0.6)].
b(X) :- X[p(Y)].
```

のようなルールがある場合、ゴール b(a) を与えると下の節の X と a, Y と 0.6 が単一化するはずだが、Y は未代入のままになる。

これでは 5.4 章で述べる `prob` や `unsure` などの実装に支障をきたす。我々は、Java で書かれた Jason の処理系のソースコードに手を加えることで、このバグを取り除いた。これで、5.4 章で述べる `prob` などをはじ

め、それを用いる *Hope* などの実装に支障をきたすことがなくなった。

5.2 実装に用いる形式化

Adam らの形式化では、感情表現の生起条件を BDI logic での論理式で表現する。例えば *Joy*(喜び) に関する生起条件は

$$Bel_i\phi \wedge Des_i\phi \supset Joy_i\phi$$

と記述される。これは「エージェント i が、事象 ϕ が生じたことを信じ、かつそれが i にとって望ましいことであるならば、i は ϕ の発生に対して喜びという感情を生起する」と読むことができる。基本的には、この式の $\supset$ の左辺が成り立つかどうかを Jason で表現し、成り立てば *Joy* という感情が生起した、と結論する。

実は、22 種の感情を 3 章の分類での①1 (a) i, ②1 (a) iiA, ③1 (a) iiB+1 (a) iiC, ④1b, ⑤2, ⑥3 の 6 種類に分けると、それぞれに属する感情同士は、生起条件の論理式がほぼ同じ形である。そこで以下では、それぞれの中から 1 つの感情について、Adam らによる生起条件の表現と、実装方針について述べる。

5.2.1 *Joy*

Joy(喜び) の生起条件は上記の通りである。$Bel_i\phi$ の表現には Jason での信念ベースをそのまま使えばよいが、$Des_i\phi$ は実は BDI モデルでいう *Desire* とは別物で、Desirability (「i にとって ϕ は望ましい」) を表すものであるため、Jason の願望 = 目標にあたる Goal を用いて表現することはできない。

Adam らの形式化では

$$Des_i\phi \iff Bel_i Des_i\phi$$

が成り立つので、$Des_i\phi$ であることと i がこれを信念に持つことは同等である。よって、$Des_i\phi$ を表現したければ、i の信念ベースにこれを入れておけばよい ($Des_i\phi$ は本来は様相オペレータを持つ式だが、Prolog の (1 階の) 項のような形式で信念ベースに保持する)。

すると、実装としては、信念ベースに ϕ と $Des_i\phi$ があるかどうかを検査し、あれば $Joy_i\phi$ を結論するという方法をとればよい。

なお、$Joy_i\phi$ を得た後、それを保持しておくには、信念ベースに追加する。正確にはこれを行うと、$Joy_i\phi$ ではなく $Bel_i Joy_i\phi$ を保持することになるのだが、*Joy* にも式 (5.2.1) と同じ性質があるので、両者は同一視してかまわない (他の感情オペレータについても同様)。このように、感情オペレータは信念ベースを用いて表現・保持することができる。

5.2.2 *Hope*

Hope(望ましい) の生起条件は

$$Expect_i\phi \wedge Des_i\phi \supset Hope_i\phi$$

で、これは「i が、ϕ が成立しそうだと思っていて、かつそれが i にとって望ましいことであるならば、i は ϕ の発生に対して期待という感情を生起する」と読む。ここで $Expect_i\phi$ は $Prob_i\phi \wedge \neg Bel_i\phi$ の略記と定義され、$Prob_i\phi$ は直感的には「ϕ である可能性が高い」のような意味である。よって $Expect_i\phi$ は「ϕ を (完全に信じてはいないものの) ありそうだと思っている」を表す。

Prob は確度の低い信念とみなせるので、Jason では annotation 付き信念で表現するのが適切である。そうすればあとは *Joy* と同様の実装が可能である。

5.2.3 *Satisfaction*

Satisfaction(満足) の生起条件は

$$Bel_i\ \mathbf{P}\ Expect_i\phi \wedge Desire_i\phi \wedge Bel_i\phi \supset Satisfaction_i\phi$$

である。「i が ϕ の成立を過去のある時に期待し (と自分が信じ)、かつ現在その成立を信じ、それが i にとって望ましいなら、i は (ϕ が達成したという) 満足感を持っている」と読まれる ($\mathbf{P}$ は「過去のある時」を表す) が、この式を実装しようとすると $Bel_i\ \mathbf{P}\ Expect_i\phi$ の部分の実現が難しい。

本研究においてはこの式を

$$Expect_i\phi \wedge Des_i\phi \supset \mathbf{A}(Satisfaction_i\phi\ \mathbf{N}\ Bel_i\phi)$$

とパラフレーズする (「i が ϕ の成立を期待し、それが i にとって望ましいなら、次に i がその成立を信じたときに i は満足する」の意。$\mathbf{A}(\psi\ \mathbf{N}\ \phi)$ は「現時刻以後最初に ϕ が成り立った時に ψ も成り立つ」を表す)。Adam らの体系では *Des* は同じエージェント i に対しては時間によって変わらないものとされる ($Des_i\phi \Longleftrightarrow \mathbf{G}Des_i\phi$ が成り立つ。ここで $\mathbf{G}$ は「未来永遠に」を表す) ため、$Des_i\phi$ を検査するタイミングはいつであってもよい。

すると、$Expect_i\phi$ を結論したらしばらくそれを保持しておき、ϕ が信念に入ったら $Satisfaction_i\phi$ を結論するとともに $Expect_i\phi$ を削除する、という実装が行える。

5.2.4 *HappyFor*

HappyFor(他者にとっての良い結果を喜ぶ) の生起条件は

$$Bel_i\phi \wedge Bel_iDes_j\phi \wedge Des_iBel_j\phi \supset HappyFor_{i,j}\phi$$

で、「i が、ϕ の成立と、j がそれを望んでいたことを信じ、また ϕ の達成を j が信じることを望む = それを j に伝えたいと望むならば、i は、j にとって ϕ が実現されたことを喜ぶ」のように読める。これは、i, j それぞれの *Bel*, *Des* に加え、Jason にエージェント間での通信があってそれらを伝えることができることを用いれば実現できる。

5.2.5 *Pride*

Pride(誇りに思う) の生起条件は

$$Bel_iDone_{i:\alpha}\xi \wedge Bel_i\phi \supset Pride_i(i:\alpha, \phi)$$
$$\text{ここで } \xi \equiv Idl_iHappens_{i:\alpha}\phi \wedge Prob_iAfter_{i:\alpha}\neg\phi$$

と書かれる。$Done_{i:\alpha}\phi$ は「i がアクション α を実行する前は ϕ が成り立っていた」、$Idl_i\phi$ は「ϕ が成り立つのは理想的な状況と見なされる (i にとって)」、$Happens_{i:\alpha}\phi$ は「i がアクション α を実行すると ϕ が成り立つ可能性がある」、$After_{i:\alpha}\phi$ は「i がアクション α を実行すると ϕ が成り立つ」を意味し、この生起条件は全体として大まかに「i が α を実行する前の時点で、i にとって α を実行して ϕ を達成するのは理想的であったが、実際には ϕ の達成は難しいと思われていた (ここまでが $Bel_iDone_{i:\alpha}\xi$)。(α を実行した後の) 現在は ϕ の

達成を信じている」を表す。すなわち、理想的だが達成は難しいと思われていたことを達成したことで、i は ϕ の達成に誇りを感じる、という特徴付けである。

この定義を実装しようとすると、ξ の部分の扱いが難しい。しかしこの部分が何らかの方法で導ければ、あとは上の式を「ξ が得られ、かつその後アクション α を行って信念 ϕ が得られれば、$Pride_i(i:\alpha,\phi)$ を結論する」とパラフレーズすることで実装できる。

Adam らの感情定義では Idl は必ず、$Idl_i Happens_{i:\alpha}\phi$ の形でしか現れない。そこで、これを i の信念ベースに $Idl\_Happens(\alpha,\phi)$ の形で持っておけば、別途 Idl オペレータを考慮することなく、同等の情報が持てる。これと、$After_{i:\alpha}\neg\phi$ の両方が得られれば ξ を結論できる。ただし、$Happens_{i:\alpha}\phi$ と $After_{i:\alpha}\neg\phi$ のいずれも、アクション α を実行した後の未来に関する予見の形の信念なので、これらをいかに得るかの考察が現時点では課題として残されている。

5.2.6 *Love*

Love と *Hate* は、様相述語論理を要するためという理由で Adam らの形式化からは省かれている。これについては単に信念ベースに保持し、*Love* や *Hate* のインスタンス毎に信念への加除条件を適当に記述するのがよいかもしれない。

5.3 実装方針

上記の各感情の生起条件を Jason の規則として表現することで、Jason におけるプランの前提条件部で使えるようにした。

今回の実験では生起され得る感情を 3 章 1 (a) iiC までに絞り、生起された感情とその過程をもとに、人間のもつ感情がこの形式化で正しいのかを検証した。

5.4 Jason における形式化の実装

5.2 章の各感情の生起条件を Jason の規則として表現し、プランの前提条件部で使えるようにした。例えば、*Joy* に関する生起条件の論理式は $Bel_i\phi \wedge Des_i\phi \supset Joy_i\phi$ のように記述されるが、これを Jason の規則として次のように記述する。

```
joy(X) :- des(X) & sure(X).
sure(X) :- X & not X[degOfCert(_)].
```

論理式通りに記述すれば `joy(X) :- des(X) & X.` となるが、本研究では「確度の低い信念」を Jason の annotation で表現しており (例えば `happy_end` を確度 0.8 で信じていることは `happy_end[degOfCert(0.8)]` で表現する)、Jason の仕様上、節の体部中の項に annotation が書かれていても annotation 付きの項と単一化してしまうため、単に「X」では X が annotation 付きである場合、即ち X の確度が低い場合を排除できない。よって、「X を確実なものとして信じている」を表す `sure(X)` という述語を導入し、「`degOfCert` という annotation を伴わない信念 X が存在する」と定義することでこれを表現した。

同様にして、*Hope* に関する生起条件の論理式は $Expect_i\phi \wedge Des_i\phi \supset Hope_i\phi$ のように記述されるが、Jason の規則としては次のように記述する。ここでは、「X を確度 0.7 以上で信じる」を表す `prob(X)` と「X を確度 1 未満で信じる」を表す `unsure(X)` を定義し、`expect(X)` をこれらの連言で定義することで「不確実だが確

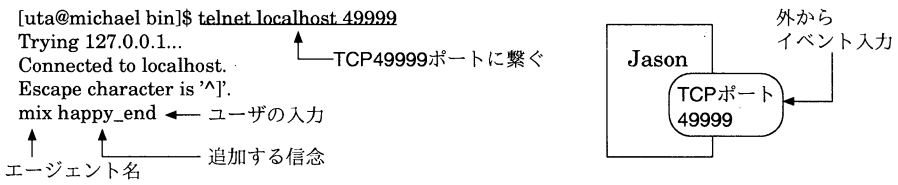

図 4　ローカルホストに接続した時の端末

```
Connected
addPercept(mix, happy_end)
```
エージェントmixにhappy_endという信念が追加された

図 5　ローカルホストに接続した時の Jason のコンソール画面

度の高い信念」を表現している。

```
hope(X) :- expect(X) & des(X).
expect(X) :- prob(X) & unsure(X).
prob(X) :- X[degOfCert(Y)] & Y>=0.7.
unsure(X) :- X[degOfCert(Y)] & Y<1.
```

6　実験と結果

本節では、実装した感情が期待通りに生起され、プラン選択に影響を与えることを実験で示す。

6.1　実験

実験としては、様々なシナリオを作り、そのシナリオ通りにエージェントを動かすことでそのシナリオに沿った感情を生起させ、それによってとる行動を変えるようなプランを作った。

またそれにあたり、エージェントに対し外から任意のタイミングで信念の追加を行うイベントやゴールの追加を行うイベントを与えてやる必要がある。ここでは、外から TCP の 49999 番ポートに接続すると、指定したエージェントにこれらのイベントを与えられるような環境を作成することでこれを実現した (図 4)。その結果、実行画面では図 5 のように表示される。

6.2　結果

6.2.1　*Joy*

Joy という感情を生起させるシナリオとしては以下のように設定する。

あるゲームに熱中しているとする。普通なら 1 人攻略できれば次へ、というようにプレイを続けていく (go_next) ところを、特に自分の気に入ったキャラを攻略でき、無事にお目当てのキャラとハッピーエンドをむかえる (happy_end) ことができたなら、嬉しいという感情が生起され、思わず歓声をあげてしまう (ktkr)。このシナリオは図 8 のようなプランで実装される。act1 というゴールが生じた際、*Joy* という感情が生起しているならば図 8 上のプラン、そうでなければ図 8 下のプランが選ばれる。

```
[uta@michael bin]$ telnet localhost 49999
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
mix happy_end ◀――― 信念ハッピーエンドを追加
mix !act1 ◀――――― ゴールact1の追加
```

図 6　*Joy* を生起させる時の操作

```
Connected
addPercept(mix, happy_end)
addPercept(mix, goal(1,act1))
[mix] !act1 ◀――――――― ゴールact1が追加された
[mix] ktkr ◀――――――― Joyが生じているため図8上のプランが
                        選ばれ、歓声をあげた
```

図 7　*Joy* を生起させた時の結果

```
+!act1
: joy(X)◀――――Joyが生成されていたら
<- .print("ktkr")◀――ktkrと叫べ

+!act1
<- .print("go next")◀―― 生成されていないなら
                        次へいく
```

図 8　ゴール `act1` に対するプラン

操作としては、起こったとする事象 (`happy_end`) を端末上でエージェントに知らせてやり、ゴール `act1` を生成させる。その際にエージェントが *Joy* を生起していれば、歓声をあげる (`ktkr`)。

この時、端末上で図 6 のように実行すると実験の結果としては図 7 のようになった。

6.2.2　*Satisfaction*

次に、より複雑な感情である *Satisfaction* を生起させるため、以下のような少し複雑なシナリオを設定する。

攻略したいと望んでいたキャラの攻略難易度 (攻略できる確率) が 8 割程度だったとする。そのキャラ攻略のために攻略サイトを閲覧したり、プレイ時間を増やしたりして攻略のために尽力する。その結果、無事攻略できたら (`happy_end`)、喜びとは別に満足感が生まれる。満足するとゲームをやめる (`stop_the_game`)。このシナリオは図 11 のようなプランで実装される。`act3` というゴールが生じた際、*Satisfaction* という感情が生起しているならば図 11 上のプラン、そうでなければ図 11 下のプランが選ばれる。

ここで、事象が起こる確率は degOfCert(0〜1 の数値) という形で表す。つまりここで言う `happy_end[degOfCert(0.8)]` は「お目当てのキャラの攻略難易度が若干低く、攻略できる確率が 0.8 である」いう意味である。このシナリオは図 11 のようなプランで実現される。

操作としては、先に 0.8 の確率で起こると予想していた事象 (`happy_end`) を信念に追加しておいてやり、それが実際に起こったことを端末上でエージェントに知らせてやる。 その際に *Satisfaction* を生起すれば、満足してゲームをやめる (`stop_the_game`)。

この時、端末上で図 9 のように実行すると実験の結果としては図 10 のようになった。

```
[uta@michael bin]$ telnet localhost 49999
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
mix happy_end[degOfCert(0.8)]   ←―― 確率0.8で攻略できると信念追加する
mix -happy_end[degOfCert(0.8)]  ←―― 上の信念を一旦削除
mix happy_end                   ←―― 実際に攻略できたという信念追加
mix !act3                       ←―― ゴールact3を追加
```

図 9　*Satisfaction* を生起させる時の操作

```
Connected
addPercept(mix, happy_end[degOfCert(0.8)])
removePercept(mix, happy_end[degOfCert(0.8)])
addPercept(mix, happy_end)
addPercept(mix, goal(1,act3))
[mix] !act3          ←―― ゴールact3が追加された
[mix] stop the game  ←―― Satisfactionが生じたため
                         図11上のプランが選ばれた
```

図 10　*Satisfaction* を生起させた結果

```
+!act3
: satisfaction(X)←――――satisfactionが生起されていたら
<- .print("stop the game").←ゲームをやめる

+!act3
<- .print("go next").←――――生起されていないなら
                         次へいく
```

図 11　ゴール `act3` に対するプラン

6.2.3　*Disappointment*

さらに、別のシナリオを考える。

普通ならバッドエンドで終了したところで、もう一度最初からゲームをやり直す (`try_again`) までであるが、自分の気に入ったキャラが攻略難易度が低かったにも関わらずバッドエンドで終わってしまう (`bad_end`) と、攻略しやすいキャラだっただけに、がっかりして食欲がなくなる (`no_appetite`)。このシナリオは図 14 のようなプランで実装される。`act3` というゴールが生じた際、*Disappointment* という感情が生起しているならば図 14 上のプラン、そうでなければ図 14 下のプランが選ばれる。

操作としては、先に 0.2 の確率で起こると予想していた事象 (`bad_end`) を信念に追加しておいてやり、それが実際に起こったことを端末上でエージェントに知らせてやる。 その際に *Disappointment* を生起すれば、がっかりして食欲がなくなる (`no_appetite`)。

この時、端末上で図 12 のように実行すると実験の結果としては図 13 のようになった。

```
[uta@michael bin]$ telnet localhost 49999
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
mix bad_end[degOfCert(0.2)]   ←―― 確率0.2で攻略できないという信念追加
mix -bad_end[degOfCert(0.2)]  ←―― 上記の信念を一旦削除
mix bad_end                   ←―― 「実際に攻略できなかった」という信念を追加
mix !act8                     ←―― ゴールact8を追加
```

図 12　*Disappointment* を生起させる時の操作

```
Connected
addPercept(mix, bad_end[degOfCert(0.2)])
removePercept(mix, bad_end[degOfCert(0.2)])
addPercept(mix, bad_end)
addPercept(mix, goal(1,act8))
[mix] !act8          ←―― ゴールact8が追加された
[mix] no appetite    ←―― Disappointmentが生じていたため
                         図14上のプランが選ばれ
                         食欲がなくなった
```

図 13　*Disappointment* を生起させた結果

```
+!act8
: disappointment(X)←―― disappointmentが生起されていたら
<- .print("no appetite").←―― 食欲がなくなる

+!act8
<- .print("try again").←―― 生起されていなかったら
                           もう一度トライする
```

図 14　ゴール `act8` に対するプラン

7　考察

これまでに述べたように、ある一定のシナリオにおいては、エージェントは OCC theory に基づいて形式化された感情を生起することができた。本節では、このシステムによる感情の生起が妥当かどうかを論じる。特に本研究での感情生起の実装は現段階では原始的なものにとどまっており、その問題点についても論じる。

7.1　複数の感情生起の競合

OCC theory における感情の形式化においては、いくつかの感情についてその生起過程が似通っているため、ある感情の生起が他の感情の生起を含意してしまうことがある。

例えば *Joy* に関する生起条件は

$$Bel_i\phi \wedge Des_i\phi \supset Joy_i\phi$$

Satisfaction に関する生起条件は

$$Expect_i\phi \wedge Des_i\phi \supset \mathbf{A}(Satisfaction_i\phi\ \mathbf{N}\ Bel_i\phi)$$

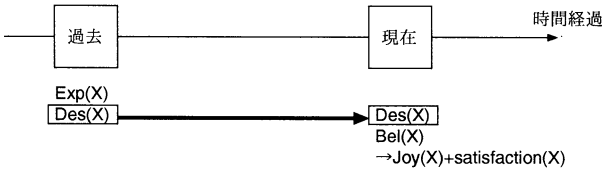

図 15　*Satisfaction* の生起過程

と表される。

これらの論理式を基にした *Satisfaction* の生起過程は以下のようになる。OCC theory における過去のある時点で *Expect* と *Desire* が生起した時、*Desire* だけは永続的に持ち続けられる。そこに *Belief* が追加されると *Satisfaction* が生起される。つまり現在の時点で *Desire* と *Belief* が同時に存在する時、*Satisfaction* が生起される。

一方 *Joy* の生起条件の論理式を見てみると、同様にして現在の時点で *Desire* と *Belief* が同時に存在する時 *Joy* が生起される。このため、*Satisfaction* が生起される時は、*Joy* も生起されてしまうことになる (図 15)。

このように同一の条件下で複数の感情が生起される場合、Jason のプラン選択に関する問題が起きることがある。Jason では、同一のイベントに対するプランが複数ある場合、(Prolog と同様に) 上に書いてあるものから順に見て、前提条件が満たされる最初のものが選ばれるため、プランは適切な順序で記述される必要がある。

例えば、6.2.1 章と 6.2.2 章で述べた `act1` と `act3` に対するプランが、同一のエージェントプログラムに書かれており、いま、*Satisfaction* が生起しているとする。この場合、同時に *Joy* も生起するので、ゴール `act1` が発生した場合、*Joy* が生起した場合のプランが選ばれてしまう。しかし、そのプランが *Joy* が単独発生した時のみを想定して書かれていたとすると、このプラン選択は適切でない可能性がある。

この問題を解決するためには、実装段階において *Joy* が生起されるプラン、*Satisfaction* が生起されるプラン、両方起こり得るプラン、というように条件分けをし、かつ上で述べた Jason の構造を考慮に入れ、「より簡単な論理式で形式化された感情」が生起された時に選択されるプランを上方に記述することで、回避できると考えられる。

7.2　複合的な感情

さらに、3 章で述べた *Disappointment* と *HappyFor* と *Reproach* のケースのように、複数の相反する感情が同時に生起されることも、人間の自然な状況として考え得る。このような複合的な感情の発生に関しては、感情それぞれに 5.4 章で述べた annotation 機能を用いることで感情の度合いを数値化し、数値の大きさによるプラン選択が行えるようにすることが考えられる。また、場合によって複合的な感情を新たに定義することも一つの方法であると考えられる。

7.3　実装上の問題点

本研究では実装しなかったが、3 章の 1 (a) iiC 以降の感情、つまり他者の感情が自分の感情に影響してくる感情を将来的に実装する際に問題になってくるのが、他のエージェントとの通信である。例えば、お目当てのキャラを攻略できなかった友達 (エージェント 1) ががっかりしているのを知り、自分 (エージェント 2) が

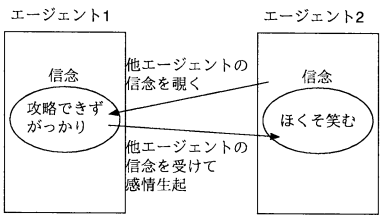

図 16　エージェント間の通信

```
+happy_end[degOfCert(Y)]
: expect(happy_end)
<- +exist_expect(happy_end).
```

図 17　変数が使えない部分

先に攻略してやろうと「ほくそ笑む」感情の生起を考える時 (図 16)、他者の感情が自分の感情に影響を及ぼすということは、他のエージェントの感情を知る必要がある。しかし Jason では各エージェントの心的状態はそのエージェント内でのみアクセスでき、他のエージェントから知ることはできないため、他エージェントの感情に依存するような感情を実装するには、他エージェントの信念が変わる度にそれをこちらに知らせることが必要になる。知らせること自体は、エージェント間の通信で実現できるが、「他者の感情を知るのに必要な情報のみを」「信念が変わる度に」知らせることをどのように実現するかが問題である。

また、図 17 は *Satisfaction* の実装において、特定の事象 (例では `happy_end`) を `expect` したら、そのことを時間が経過しても覚えておく必要があるため、「過去に `happy_end` を `expect` した」を表す `exist_expect(happy_end)` という信念を保持しておくというものであるが、同じ処理は他の事象においても必要であるため、本来はこのプランの「`happy_end`」の部分を変数にして、一般化すべきである。しかし、プランのトリガイベント自体を変数とすることは構造上不可能であるため、現実にはこの処理を必要なだけたくさん書かねばならないという問題がある。

7.4　*Surprise*(驚き) という感情

通常我々は、人間の感情のうちの重要な一つとして「驚き」を考える。しかし、OCC theory では *Surprise* は感情として扱われていない。その理由は驚きにはプラスの意味での「嬉しい驚き」とマイナスの意味での「悲しい驚き」とがあり、それぞれが 22 種類の感情のいずれかに分類されるため、「驚き」自体は感情には含まれないとされていることである。

しかし本研究では、実際の人間の感情として驚きは必要不可欠であるため、*Surprise* を感情として定義することは重要であると考える。

例えば、5 章で述べたように、*Satisfaction*(満足) と *Relief*(安堵) は時間的変化を伴う感情である。ここで、*Satisfaction* が生起されるまでの間に $Prob_i\phi$ が消失する可能性について考える。つまり過去の望みが、達成されるまでに消えてしまった (望みがなくなった) 場合、それにも関わらず望んでいた事が実現したとする。その場合に生起される感情は単なる「満足」と定義してしまうのは妥当であろうか (図 18)。これは単なる満足ではく、「望みがないと思っていたことが予想に反して実現した」ことへの「驚きを伴った満足」であると言えるのではないだろうか。

では、その「驚き」という感情はどのように定義すべきなのか。一つの方法としては、`desire`(望み) や

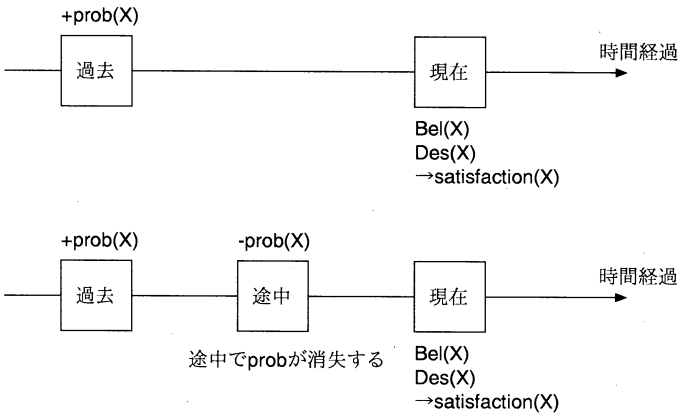

図 18　上段と下段の「満足」の違い

expect(期待値) に度合いを付加することで、変化の振り幅が表現できるため、これによって *Surprise* が表現できると考えられる。つまり、元の desire や expect の程度が低いほど、それが実現した時の意外性が大きいと言え、即ちそれが驚きに繋がる。よって驚きという感情が表現できると考えられる。今後の課題の一つである。

8　終わりに

本研究では、感情をすべて BDI エージェントの信念として扱うことで、BDI ロジックに形式化された感情を、既存の Jason インタプリタを用いて実装した。さらに BDI エージェントで感情を生起させ、その感情により行動決定をする実験を行い、期待通りの行動決定が行われた事と、現状での問題点について述べた。今後は、現在まだ実装していない感情の実装や 7 章で述べた感情の度合いの導入、複合的な感情への対応などを行っていき、より自然な人間に近い感情を生起できるエージェントを目指す。

謝辞

本研究を進めるにあたり、指導教官の新出尚之准教授、並びに博士後期課程藤田恵先輩から丁寧な御指導・助言を賜りました。心から感謝の気持ちと御礼を申し上げたく、謝辞にかえさせて頂きます。

また、多くの知識や示唆を頂き支えてくださった、新出研究室の先輩方に感謝致します。

参考文献

[1] Carole Adam, Andreas Herzig, and Dominique Longin. A logical formalization of the OCC theory of emotions. *Synthese*, Vol. 168, No. 2, pp. 201–248, 2009.

[2] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason.* WILEY, 2007.

[3] Patricia Augustin Jaques and Rosa Maria Vicari. A BDI approach to infer student's emotions in an intelligent learning environment. *Comput. Educ.*, Vol. 49, pp. 360–384, September 2007.

[4] A. Ortony, G.L. Clore, and A Collins. *The Cognitive Structure of Emotions.* Cambridge University

Press, 1988.

[5] David Pereira, Eugénio Oliveira, and Nelma Moreira. Modelling emotional BDI agents. In *Workshop on Formal Approaches to Multi-Agent Systems (FAMAS 2006), Riva Del Garda*, 2006.

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

[7] 藤田恵, 片山寛子, 新出尚之, 高田司郎. 実世界の多様性に適応した BDI ロボットについて. 情報処理学会論文誌 数理モデル化と応用, 2012. to appear.

[8] 黒宮寧, 礒田佳徳, 長沼武史, 稲村浩, 倉掛正治. 感情を伝え合うコミュニケーションを行うインタフェースエージェント. 情報処理学会研究報告ヒューマンコンピュータインタラクション (HCI), Vol. 2004, pp. 33–40, 2004.

[9] 傳刀洋輔, 亀井且有. 意思伝達に感情を用いるエージェントシステムの構築. 第 18 回ファジィシステムシンポジウム, pp. 381–384, 2002.