論　説

ビジネスゲームのための言語の設計と実装

田名部　元成

1．はじめに

YBG（Yokohama Business Game）は，ビジネスゲームの開発と実施，および運用を支援するためのシステムであり，2001年頃から白井宏明を中心とする横浜国立大学経営学部のグループによって研究開発されてきたものである．YBGの中核部分は，筑波大学で開発されたビジネスゲーム生成システムBMDS（Business Model Development System）とビジネスゲーム記述言語BMDL（Business Model Description Language）という二つのコンポーネントである（白井・藤森ほか，2000；Tsuda et. al, 2002）．BMDSは，BMDLで記述されたビジネスゲームのモデルからビジネスゲームの実施環境をウェブサーバ上に自動生成するシステム，すなわちビジネスゲームのジェネレータである．

BMDSを用いたビジネスゲームの実施環境の生成には，通常，ゲーム開発者が仮想端末ソフトウェアによってウェブサーバにログインし，その上でBMDL変換プログラムを起動する必要があった[注1)]．ウェブサーバへのリモートアクセスを許可することは，サーバの脆弱性を増大させることにつながるため，このようなアクセスはイントラネットからのアクセスに限られるのが通常である．YBGは，BMDSのこのような制約を取り除き，ゲームの開発や開発したゲームの管理をリモートサーバへのログインなしに，標準的ウェブブラウザのみを通じて行えるようにしたものである．

YBGの利用環境のウェブ化は，ゲームの実施機会と開発機会を大幅に増大させることにつながった．その結果，ビジネスゲームの開発面と運用面のさらなる強化が必要となり，2003年頃からバージョン２シリーズの開発が始まっている[注2)]．そして，YBG2.0では，ビジネスゲームのモデル記述に関してはBMDLの言語仕様を踏襲しながらも，その処理方法に関してはBMDSとは異なる方法でビジネスゲームの実施環境を生成するようになる．

YBGを活用した教育実践を含む取り組みは，平成16年度文部科学省「現代的教育ニーズ取組支援プログラム」（2004 ～ 2006年度），平成19年度文部科学省「特色ある大学教育支援プログラム」（2007 ～ 2009年度）に採択され，その間YBGは，ビジネスゲームの教育実践の結果に基づいて，バージョン3.0，3.1，2007，2008，2009シリーズと順次開発されていく．2010年以後は，幾つかの開発プロジェクトにより改良が加えられ，現時点の最新はYBG2011シリーズである．これら一連のYBGシステムの言語処理の仕組みは，YBG2.0を基盤として再構築されたYBG3.0

のそれと同じである．

YBGシリーズは，ビジネスゲームの開発，運用，実施といった一連の作業を念頭に置いたウェブベースの統合的ビジネスゲーム支援システムである．近年では，eラーニングプラットフォームやSNS（Social Networking Service）など，外部のシステムやサービスとの連携が求められる場面や，より高度なビジネスゲームの開発ニーズも出現してきており，ビジネスゲーム支援システムは，学習管理システムの教育現場への普及，モバイルコンピューティングやクラウド環境の一般化，学習者ニーズの多様化，より高度な経営教育手法の開発と実践の必要性など，近年の情報技術の利用動向や利用者環境の趨勢を踏まえた，ウェブサービスへと進化する必要が求められている．

本論では，ビジネスゲームを支援するためのウェブサービスに求められるビジネスゲームの実行系に焦点を当てる．まず，複数の意思決定主体が同じ立場で，特定の市場でビジネスを行うという典型的なビジネスゲームをオートマトンとして定式化し，ビジネスゲームの実行系を，プレイヤのプロセスと市場のプロセスという部分システムからなる入出力システムとして表現する．そして，ビジネスゲーム実行系が，クラウド上に実装される際のアーキテクチャについて考察する．続いて，ビジネスゲーム実行系の核となる言語処理系の設計に必要となるゲームモデルにおける変数の取り扱いについて考察し，モデル変数間の関係性を計算処理するための仕組みについて述べる．本論では，この考え方に基づく言語処理系の実装ついても述べる．最後に，この処理系を用いたいくつかのビジネスゲームの実装と利用事例を紹介する．

２．ビジネスゲームのオートマトンモデル

複数の意思決定主体が同じ立場で，特定の市場でビジネスを行うという典型的なビジネスゲームを因果的な離散時間定常システムとしてモデル化する．いま，数量や金額などを表す集合として実数全体の集合Rを仮定する．YBGやBMDSでは，伝統的にビジネスゲームにおける企業や組織などの意思決定主体をチームと表現する．これは，ビジネスゲームで競う主体が，複数人で構成されていることを前提としているからである．また，ゲームの進行役をコントローラと呼ぶ．本論でも，これにならってゲームの中で競う企業や組織などの意思決定主体の単位としてチームを用い，ゲーム進行役のことをコントローラと呼ぶことにする．

ゲームにn組のプレイヤチームが意思決定主体として参加し，各チームがm個の項目を意思決定するものとする．このとき全チームの意思決定は

$$x_P=\begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}\in M(m,\ n;R) \tag{1}$$

と表される．ビジネスゲームには，コントローラの意思決定（制御）が入力として与えられる場合もある．このようなコントローラからの入力を$x_M\in M(m_M,\ 1;R)$とする．また，ゲーム外部からの外乱入力を$x_U\in M(m_U,\ 1;R)$とする．すると，ゲームへの入力は，

$$x=(x_P,\ x_M,\ x_U)\in A_P\times A_M\times A_U \tag{2}$$

のように，３つ組として表される．ただし，$A_P = M(m, n; R)$, $A_M = M(m_M, 1; R)$, $A_U = M(m_U, 1; R)$である．

ゲームの状態は，各チームの状態と市場などのチーム外部環境の状態の組として表される．各チームに，l個の状態変数があるとすると，全チームの状態は，

$$z_P = \begin{pmatrix} z_{11} & \cdots & z_{1n} \\ \vdots & \ddots & \vdots \\ z_{l1} & \cdots & z_{ln} \end{pmatrix} \in M(l, n; R) \tag{3}$$

と表される．チーム外部環境の状態を$z_M \in M(l_M, 1; R)$とすると，ゲームの状態は

$$z = (z_P, z_M) \in C_P \times C_M \tag{4}$$

と表される．ただし，$C_P = M(l, n; R)$, $C_M = M(l_M, 1; R)$である．

状態遷移関数δ: $(C_P \times C_M) \times (A_P \times A_M \times A_U) \to C_P \times C_M$は，プレイヤと市場の状態遷移関数

$$\delta_P : (C_P \times C_M) \times (A_P \times A_M \times A_U) \to C_P, \tag{5}$$
$$\delta_M : (C_P \times C_M) \times (A_P \times A_M \times A_U) \to C_M. \tag{6}$$

によって，$\delta = (\delta_P, \delta_M)$と書くことができる．ただし，任意の$c \in C_P \times C_M$, $a \in A_P \times A_M \times A_U$に対して$\delta(c, a) = (\delta_P(c, a), \delta_M(c, a))$である．

出力関数λ: $(C_P \times C_M) \times (A_P \times A_M \times A_U) \to B_P \times B_M$も同様に，

$$\lambda_P : (C_P \times C_M) \times (A_P \times A_M \times A_U) \to B_P, \tag{7}$$
$$\lambda_M : (C_P \times C_M) \times (A_P \times A_M \times A_U) \to B_M. \tag{8}$$

によって，$\lambda = (\lambda_P, \lambda_M)$と書くことができる．ただし，$B_P$, B_MはR上の適当な行列であり，任意の$c \in C_P \times C_M$, $a \in A_P \times A_M \times A_U$に対して$\lambda(c, a) = (\lambda_P(c, a), \lambda_M(c, a))$である．

３．ビジネスゲームシステムの構造化

ここでは，ビジネスゲームを複数のサブシステム（プロセス）から成るものとして構造化する．図１は，チームが同じ立場で，特定の市場でビジネスを行う典型的なビジネスゲームの構造を示している．図中のチームプロセスは，複数のチームを略記したものであり，したがってチーム意思決定は，各チームの意思決定すべてを表している．チームプロセスの出力で市場プロセスへの入力となっているものは，たとえば販売価格や広告費，あるいはそれらから計算される価格競争力や広告効果などを示している．市場プロセスへの外部入力の典型的な例は，総需要である．チームへの外部入力は，各チームやチーム全体に与えられるゲームシステムへの外乱を表しているが，通常のビジネスゲームでは，省略される．外部入力は，ビジネスゲームの設計者が事前に定義する場合や，外部のデータベースや実際の市場のインデックスから計算される場合がある．市場プロセスから，チームプロセスへの入力は，典型的には，市場占有率や受

注数などが例として挙げられる. また, 市場プロセスやチームプロセスからの出力は, コントローラや各チームに提供される．コントローラ入力は，ゲームコントローラによるゲーム途中の動的な制御を意味する．だだし，コントローラ入力は，多くのビジネスゲームでは用いられない．

図１でのチームプロセスからの出力は，市場プロセスへの入力となり，市場プロセスからの出力は，またチームプロセスへの入力となっているが，このような表現は，処理の順序関係に曖昧性をもたらす．この曖昧性は，図２のようにチームプロセスを構造化することで解消することができる．図２のT1は，市場プロセスへの入力となる出力を与え，T2は，市場プロセスからの出力に基づいてチームの内部プロセスを処理する．

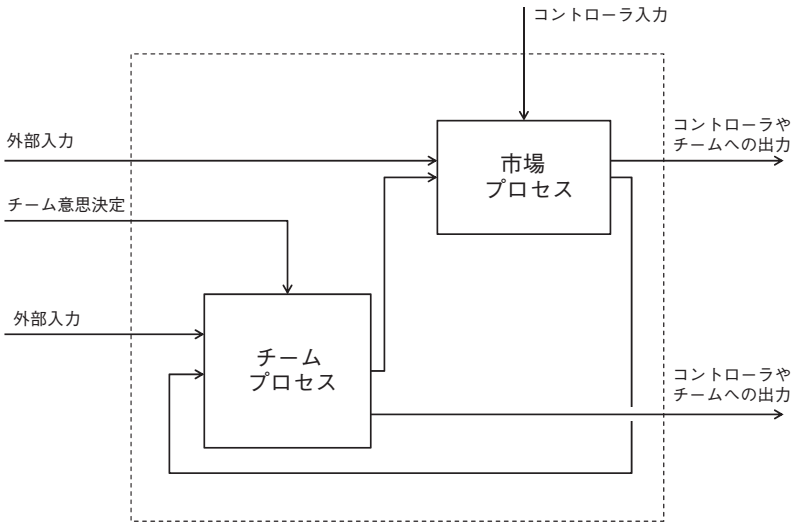

図１　入出力システムとしてのビジネスゲーム

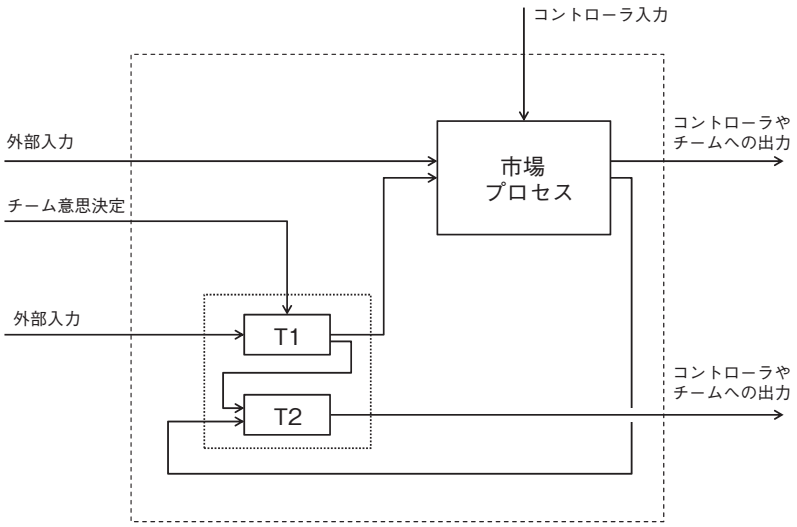

図２　構造化されたチームプロセス

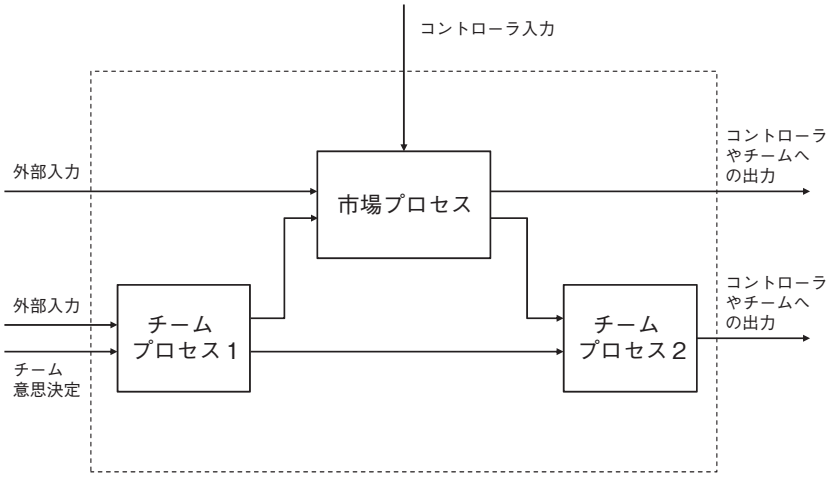

図３　図２と同じ構造のビジネスゲームシステム

図２をさらに簡潔に表現したものが図３である. 上述した通り, チームプロセスは, 複数のチームのそれぞれのプロセスを多重化してひとつとして表現している．同様に市場プロセスも複数の市場が多重化されたものと見ることもできる.

４．クラウド型ビジネスゲーム処理システム

前節で述べたビジネスゲームシステムをコンピュータ上で動作させるには，外部入力，チーム意思決定，コントローラ入力といったシステムへの入力を受け取り，市場プロセスやチームプロセスの内部状態に基づいて，状態遷移と出力を処理する仕組みを構築すればよい．システムに与える入力がエージェントによって順次計算されるようなエージェントベースシミュレーションを実装する場合，十分なメモリ空間があるならば，市場プロセスやチームプロセスの状態を初期状態から順次蓄積していくことで高速な処理を実現することが可能である．しかしながら，人間がチームプロセスへの入力を与えていくようなゲーミングシミュレーションを実装する場合，ビジネスゲームの処理系は，市場とチームの状態をファイルシステムやデータベースから読み出し，それらの次の状態と出力を計算しなければならない．言いかえれば，必要な過去の状態データと入力データを，状態遷移と出力を処理するプログラムに引き渡して，それぞれのデータを生成しなければならない.

ウェブサイトやクラウド上のウェブサービスとして，ビジネスゲーム実行系を実装する場合，クライアントの要求によって起動されるプロセスは，メモリ上に常駐しないため，起動時にゲーム状態を読み込んで，モデルで規定される計算処理を行い，新しいゲーム状態をデータベース等に書き込んで終了する必要がある．図４は，クラウドやウェブサーバにおけるゲーム実行系のアーキテクチャを示したものである．このアーキテクチャは，図１－３に示されるビジネスゲームの入出力システムモデルに基づく．この実行系では，外部データ入力プログラム，コントローラインタフェース，チームインタフェースの３つの標準サブシステムを有し，それぞれデータベース管理システム（DBMS）を通じて, 各種のデータベースにデータの読み書きを行う.

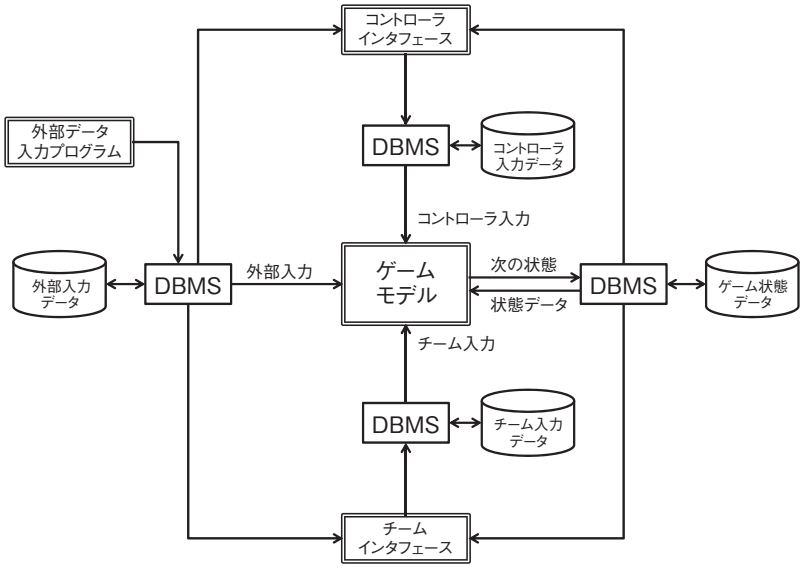

図４　クラウド型シミュレータのアーキテクチャ

コントローラインタフェースや，チームインタフェースは，これらを通じてデータ入力や出力表示をモバイル機器上のアプリケーションや別のウェブアプリケーション行うことを想定しており，ユーザが直接操作するためのユーザインタフェース，あるいは，知的エージェントや他のプログラムが操作するためのプログラミングインタフェースを意味する．

５．ビジネスモデル記述言語BMDL

YBGやBMDSが想定するビジネスゲームは，仮想的市場にチームが同等な立場で参加し，販売価格や材料調達数などの意思決定を何期かに渡って繰り返し行ない，利益最大化や費用最小化などを競うというものである．一般のビジネスゲームでは，同一チーム内でも，生産部門や財務部門などのように，異なる役割ごとに異なる意思決定が求められるものや，サプライチェーンのようにチームが市場において異なる役割を持ち，したがって異なる意思決定が求められるもの，あるいは，ゲーム進行が幾つかの段階に分けられており，それぞれの段階によって異なる意思決定が求められるものなどがあるが，YBGやBMDSが処理するBMDLは，ゲームモデルの設計の分かりやすさや処理のしやすさを優先させ，様々の可能性を単純化している．ここでは，BMDL（久野，2001）における変数の考え方と変数間の関係性を実現するための処理方法について述べる．

YBGやBMDSの前提とするビジネスゲームでは，ゲームは離散的な時間（期やラウンドと表現される）に沿って進行する．モデル構成要素は，変数として表現され，各変数間の関係は，幾つかの命令と代入式で表現される．上述した通り，意思決定主体のチームは，同じ立場や役割でゲームに参加するため，チームに関係するモデル構成要素は，同じ変数で表現される．たとえば，チームのある期の期末の売上高を表わす変数「売上高」は，rを期番号，tをチーム番

号とするとき，論理的には，売上高[r][t]のように２次元配列として表現される．BMDLでは，このような変数をチーム変数と呼ぶ．

一方，モデルを構成する要素の中には，毎期の総需要や，消費税率のようにチームに依存しないものもある．総需要は，ゲームの前期の状態をもとに決定される場合や，ゲーム設計者によって事前に定義される場合，あるいは，特定の確率分布に従う変数にもとづいて決定される場合がある．BMDLでは，期毎に値を有するがチームによらない変数をシリーズ変数[注3)]と呼び，とくに，その値が変化しないものをシリーズ定数と呼ぶ．たとえば，ゲーム設計者が事前に各期の総需要を定めるような場合は，シリーズ定数として「総需要」を宣言し，各期の値を定義することになる．シリーズ変数あるいはシリーズ定数としての「総需要」は，rを期番号とするとき，論理的には，総需要[r]のように１次元配列として表現される．シリーズ定数とシリーズ変数のモデル作成における実用上の違いは，シリーズ定数は，一度定義しておけば，各期の諸変数の値の計算において，その定数をそのまま用いることが可能であるのに対して，シリーズ変数の場合は，期首での初期化（変数への代入）が必ず必要となる点である．

期やチームによらず，ゲームを通して変化しないようなモデル構成要素は，BMDLでは，広域定数と呼ばれる．シリーズ定数に対するシリーズ変数の関係と同じように，広域定数に対して，広域変数というものも考えることもできる．たとえば，消費税率が最初は５％に設定されているが，ゲームの途中で何らかの条件によって別の値に変化させる場合が，これに相当する．

BMDLでは，値がモデル中で定義される変数や定数のほかに，外部から与えられる変数として入力変数とコントローラ入力変数が定義されている．入力変数とは，チームが期毎に意思決定してシステムに与える入力であり，同様にコントローラ入力変数とは，コントローラによるシステムに対する制御変数を表す．BMDLにおける変数と定数の一覧を表１に示す．

表１　BMDLにおける変数と定数

変数・定数	期に依存	チームに依存	動的な値変化	外部入力	説明
広域定数	×	×	×	×	「製品最低価格」など
シリーズ定数	○	×	×	×	「総需要」など
シリーズ変数	○	×	○	×	「全チーム売上合計」など
チーム変数	○	○	○	×	「売上高」など
入力変数	○	○	○	○	「販売価格」など
コントローラ入力変数	○	×	○	○	「総需要」など

チームが同一の構造を有するようなビジネスゲームのモデルでは，各変数が，期とチームに依存するかどうか，その変数の値が，外部から与えられるものであるかどうかによっても分類することができる（表２）．

表２において，期に依存しない外部から与えられる変数（チーム固有入力定数，広域入力変数）は，実装上，期に依存する対応する変数（チーム変数，シリーズ入力定数）で代用可能である．表１に示されるBMDLの変数のうち，対応するものが表２にないものは，シリーズ定数である．前節でみたとおり，クラウド上で実現するビジネスゲーム処理系は，状態データと入力データを取得し，次状態を生成するというものであった．このような処理系では，各期の総需要のような時系列データは，モデルの中で定義するよりも，シリーズ変数として，外部データやコン

表2 モデル変数の分類

番号	変数名称	期に依存	チーム に依存	外部 入力	説明
1	チーム 入力変数	○	○	○	各プレイヤが毎期，意思決定する項目．「販売価格」，「製品製造個数」など．BMDLにおける入力変数に対応．
2	チーム変数	○	○	×	各プレイヤの経営状態を示す項目．「売上高」，「製品在庫数」など．MBDLにおけるチーム変数に対応．
3	シリーズ 入力変数	○	×	○	コントローラや外部データによって定まるチーム非依存の項目．「総需要」など．BMDLにおけるコントローラ入力変数に対応．
4	シリーズ 変数	○	×	×	個別のチームによらない，市場全体の状態を表す項目．「全チーム売上合計」，「市場成熟度」など．BMDLにおけるシリーズ変数に対応．
5	チーム固有 入力定数	×	○	○	コントローラや外部データによって個々のチーム毎に定まる項目．チームの「格付け」など．BMDLの仕様にはない．
6	チーム固有 定数	×	○	×	期によらず変動しない，チーム固有の定数．「会社名」など．BMDLの仕様にはない．
7	広域 入力変数	×	×	○	コントローラや外部のデータによって動的に定める市場の特性などを表す項目．「税率」など．BMDLの仕様にはない．
8	広域変数	×	×	×	市場や企業などの構造パラメタ．「最低販売価格」，「価格弾力性」など．BMDLにおける広域定数に対応．

トローラ入力によって定義し，必要なデータのみを読み込んだ方が，他のデータの処理との整合が図られる．

BMDLにおいて，各種の変数の値の書き換えは，tlet命令を用いて行う．例えば，販売価格と販売数量から，売上高を計算するには，

```
tlet 売上高 = 販売価格 * 販売数量
```

と記述する．tlet 命令は，各チームに対して，それぞれの所有する変数に対して同じ計算を行う．BMDSやYBG1.0では，上記の記述は，内部で

```
for (t=0; t < MAXT; ++t) {
        売上高[r][t] = 販売価格[r][t] * 販売数量[r][t];
}
```

というC言語コードに変換される[注4)]．ここで，tはチーム番号，rは期番号を表す．もし tlet命令の中に，シリーズ定数が現れる場合には，例えば，「総需要」は，総需要[r]という1次元配列に変換される．このように，変数からC言語コードへの変換は，変数名と変数型を管理する

変数表に基づいて行われる．変数表には，変数名とその変数の型（チーム変数やシリーズ定数など）が格納されており，BMDSは，モデル記述において変数名が現れるたびに，変数表を探索し，その変数型に応じた内部表現へと変換する．

6．DSLによるシミュレーション処理

特定の言語によって書かれた記述を処理するためには，言語仕様を設計し，それに基づいて字句解析や構文解析を行う言語処理系を実装するというのが通常のやり方である．しかしながら，ビジネスゲームを記述するのに，個人が利用する範囲で，言語を設計し，言語処理系を実装するのはあまり現実的ではない．一方，利用者を想定して言語仕様の設計と処理系の実装を行うには，細心の注意と多くの動作確認が必要となる．頻繁に言語仕様を変更することは，利用者を混乱させるだけでなく，過去の資産の活用を不可能にする．

本論では，汎用プログラミング言語を用いて，ビジネスゲーム用のドメイン固有言語（Domain Specific Language; DSL）を実装するアプローチによって，上述の問題に対処する．ドメイン固有言語とは，特定の領域の問題を処理，解決するためのプログラミング言語である．ここでは，ビジネスゲーム設計という固有領域のための言語を，汎用プログラミング言語上に実装する．このような言語を実装する際の最大の問題は，変数と演算子から構成される式をどのように評価するかである．「売上高 ＝ 販売価格 * 販売数量」という文字列から「売上高」「＝」「販売価格」「*」「販売数量」というトークンの並びに分解して，構文を解析し，計算を行うのは，上述した構文解析アプローチと同じであり，したがって，言語処理部分を実装する必要がある．

もし，「売上高 ＝ 販売価格 * 販売数量」という表現，あるいは，その類似表現が，DSLが実装されている汎用プログラミング言語で直接処理されるならば，パーサやコンパイラの実装を必要としないため，処理が簡単になる．ただし，汎用のプログラミング言語とは異なり，変数への代入処理は，チームすべてに対して実行されなければならないため，「売上高 ＝ 販売価格 * 販売数量」を構成する変数が，文脈に応じて，それぞれ対応する異なる実体を参照しなければならない．もし，「売上高」や「販売価格」，「販売数量」が値でなく，ポインタや参照である場合には，間接参照あるいはデリファレンスを用いて，参照先の実体にアクセスすることができる．このことは，C言語的な表現では，

```
*sales = *price * *sold;
```

と書ける．このsales，price，soldに，特定のチームの「売上高」，「販売価格」，「販売数量」が保持されているメモリ上のアドレスをそれぞれ代入した後に，上の式を実行すれば，そのチームの「売上高」を計算することができる．Perl言語の書き方では，

```
${$sales} = ${$price} * ${$sold};
```

と表される．ここで，$sales，$price，$soldは，「売上高」，「販売価格」，「販売数量」への参照（reference）である．Perlでは，参照を表す変数を${ }で囲むことによって，参照先の実体を表すことができる．これはデリファレンス（dereference）と呼ばれる．

このように，参照を利用する方法によって，チーム変数を取り扱う場合，実体としての変数と，参照としての変数の２種類の変数を定義する必要がある．例えば，10チームに対してチーム変数「売上高」を定義する場合，売上高の値を記憶するための10個のメモリ領域，すなわち実体としての「売上高」(${$sales})と，指定されたチームの「売上高」の参照を格納するための「売上高」($sales)の２種類が必要となる．この方法は，チーム変数のみならず，BMDLにおける広域定数やシリーズ定数にも適用可能である．広域定数については，実体としての１つの変数と，参照としての１つの変数を用意すればよく，シリーズ定数については，期番号に応じた実体を表す変数と，参照を表す変数を用意すればよい．

BMDLでは，変数の相対的な過去の値を参照するために「@」という演算子を用いる．例えば，「期末在庫@1」は，１期前の期末在庫を意味する．gg7やYBGでは，@nは配列要素を表すインデックスに変換されて処理される．上述のデリファレンスの手法を用いるのであれば，@を演算子として扱わず「期末在庫@1」そのものを変数として扱うことも可能である．またPerlの表現で，$期末在庫@nが，あるチームのｎ期前の期末在庫への参照を表すならば，この変数の値を返す関数「期末在庫(n)」を実装して，モデル変数を取り扱うことも可能である．この場合，「期末在庫(n)」は，n期前の期末在庫への参照を表し，したがってPerlの書き方では，

```
${期末在庫(0)} = ${期末在庫(1)} + ${入庫数(0)} - ${出庫数(0)};
```

という記述が可能となる．また，「期末在庫()」という引数なしの関数呼び出しを，「期末在庫(0)」と同じ処理とするようにすることで，

```
${期末在庫()} = ${期末在庫(1)} + ${入庫数()} - ${出庫数()};
```

と書くことができる．この表記に対応するBMDLにおける記述は

```
期末在庫 = 期末在庫@1 + 入庫数 - 出庫数
```

となり，MBDLと同様の記述が，本論で提案するDSLでも記述できることを示している．

７．シミュレーション実行系の実装

ここでは，前節で述べたデリファレンスの方法を利用したビジネスゲームのためのDSLとその処理系の例として，著者がPerlで実装したBSimモジュールを紹介する．BSimモジュールはBSimAgent, BSimLang, BSimDataという３つのパッケージからなり，ビジネスゲームの動作を記述したPerlプログラムから利用される．それらの依存関係を図５に示す．図のゲーム本体は，シミュレーションの構造パラメタなどを定義した設定ファイル(conf.pl)を読み込み，言語処理用パッケージBSimLangとデータ入出力用パッケージBSimDataを読み込む．ゲームのモデル変数の宣言や動作の定義は，BSimLangで定義された関数の呼び出によって行われる．なお，BSimAgentは，BSimLangを通じて呼び出され，チームや市場などのエージェント（オブジェクト）を生成するのに用いられる．BSimAgentクラスのオブジェクトは，生成時には特別な構

造はほとんど持っていないが，ゲーム本体でチーム変数が定義されるたびに，チーム変数が属性として追加されていく．たとえば，ゲーム本体で，

```
tvar 受注数 =>  4;
```

と記述すると，これはtvar(受注数=>4)と解釈され，BSimLangで定義されているtvar関数に，キー「受注数」の値が4であるとして引き渡される．BSimLangでは，チーム変数の参照を格納するハッシュ[注5)]に「受注数」というキーを追加する．そして，生成されているチームオブジェクトすべてに対して，「受注数」という変数を追加し，その値を4に設定する．そして，本体プログラムからアクセスできる「受注数()」という関数を動的に生成する．したがって，tvar関数でチーム変数を定義した後は，その変数名と同じ名前の関数を用いることができる．

変数への値の代入には，BSimLangで定義されているtlet 関数を用いる．tlet 関数は，コードと処理対象エージェント集合を引数にとり，処理対象エージェントすべてに対して，引き渡されたコードを実行する．一般形は，

```
tlet { コード } エージェント集合
```

または，

```
tlet { コード }
```

である．たとえば，

```
tlet {
        ${累積販売個数()} = ${累積販売個数(1)} + ${販売個数()}
} with { TEAM % 2 == 0 };
```

という記述は，チーム番号が偶数であるチームに対して，累積販売個数を計算することを意味

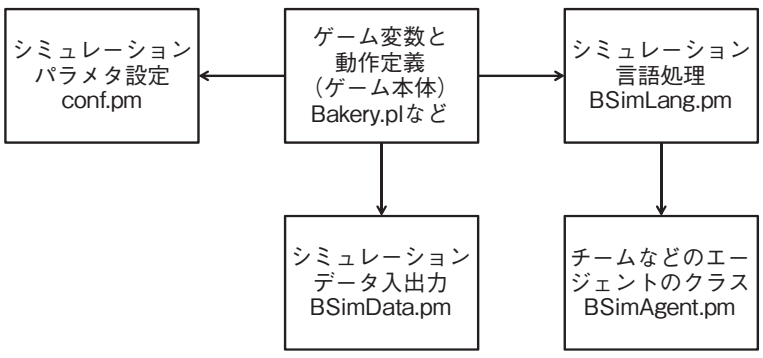

図５　BSimモジュールとゲーム定義の依存関係

する．tlet 直後の { }部分がコード部分であり，with以降がエージェント集合に対応する．なお with もBSimLangで定義されている関数で，条件式（コード）を引数にとり，その条件が満たされるエージェント集合を返す．tlet関数において，エージェント集合は省略可能で，もし省略された場合は，すべてのチームに対して，指定されるコードが実行される．

前節で述べたとおり，tlet関数におけるコードは，参照を返す関数のデリファレンスによって処理される．tletのBSimLangにおける実装は，次の通りである．

```
sub tlet(&@) {
    my ( $code, @agentset ) = @_;
    @agentset = @agents if !@agentset;

    for my $obj (@agentset) {
        _set_param($obj);
        $_ = $obj;
        $code->();
    }
}
```

ここで$objは，各チームオブジェクトへの参照である．_set_param関数は，チーム変数以外の変数について，当該期および過去の値への参照を，ゲーム本体のプログラムで利用可能な変数に代入し，また，引数で指定されたチーム$objの当該期および過去の各チーム変数への参照をゲーム本体プログラムで利用可能なチーム変数に代入する．そうすることで，引数で与えられた関数への参照$codeを実行するたびに，$objの各チーム変数の代入計算を行うことが可能となる．例えば，tlet関数の呼び出し

```
tlet { ${累積販売個数()} = ${累積販売個数(1)} + ${販売個数()} }
```

は，

```
$code = sub {
    ${累積販売個数()} = ${累積販売個数(1)} + ${販売個数()}
};
$code->();
```

という処理を各チームオブジェクトに行うものと解釈される．このコードは，通常のサブルーチンの書き方に直せば，

```
sub code {
    ${累積販売個数()} = ${累積販売個数(1)} + ${販売個数()}
}
code();
```

となる．

Perlでは，関数への参照$codeが$code->()として実行されるとき，引数以外の変数を関数定義時の環境で解決することが可能である．このように，関数の引数以外の変数の解決を，関数実行時ではなく関数定義時の環境によって行う関数は，クロージャ（closure）と呼ばれる．BSimLangでは，BSimLang内の他のサブルーチンが，tletを呼びだす際に，変数$_を参照することを想定している．そのため，例えば，$code->()を呼びだす前に，$_ = $objを実行することにより，関数$code->()の実行時に，チームオブジェクト$objへアクセスすることが可能となる．

8．DSLの利用例

本節では，BSimモジュールを用いた，シミュレーションの利用例を示す．最初の事例は，ビールゲームのシミュレーションである．ビールゲームは，1960年代初期にMITで開発された生産流通におけるシステムダイナミクスを学ぶためのゲーミングシミュレーションである(Sterman, 1992)．現在では，サプライチェーンに対する理解を深めるための教育ツールとして国内外問わず広く使われている．このゲームでは，小売店，二次卸，一次卸，工場の４段階から構成されるサプライチェーンを最低４人からなるプレイヤチームで運営し，50期にわたって各段階における発注量（生産量）を毎期決定しながら，サプライチェーン全体における費用最小化を目指して競い合う．製品は上流（工場）から下流（小売店）へと移動し，注文情報は間接的に下流から上流へと移動する．費用は在庫費用と受注残費用の合計であり，発注や配送にはリードタイムがある．また注文情報は送付先以外のプレイヤは参照することが出来ず，異なる段階にいるプレイヤ同士のコミュニケーションは許されないという実施上のルールがある．

BSimモジュールを利用して，このゲーム構造を実装した例が図６である．注文数の決定には，様々な在庫管理手法が適用できるが，BSimでは，外部のデータを直接読み込んで利用することも可能である．図７は，実際のビールゲームの実施結果を外部データとして用意し，それをシミュレーション実行時に意思決定値として読み込みながら行ったシミュレーションの結果である．図において「在庫」の括弧内は，在庫数と受注残数の組を表す．このシミュレーションモデルに統計量を表す変数を導入することにより，実際の意思決定を様々な角度から分析することが可能となる．

２つ目の利用例は，ベーカリーゲームにおける事後分析への活用である．ベーカリーゲーム（白井，2008）は，10名程度のプレイヤが，パンの製造販売を行いながら，営業利益を競うゲームである．各プレイヤは，毎期，パンの材料調達個数，製造指図数，販売価格の３つの項目を意思決定する．材料納入と製品製造のリードタイムはそれぞれ１期である．各プレイヤに対する需要（顧客数）は，各プレイヤの決定した販売価格に応じて，総需要が分配される形で決定される．品切れは次期以降の需要に影響を与え，売れ残りは廃棄損失となる（田名部，2011a）．

図８は，BSimモジュールを用いて実装した，ベーカリーゲームの主要プロセスを示している．すべてのサブプロセスは，サブルーチン化されており，各サブルーチンの中で，チーム変数への代入処理などが行われる．例として，生産プロセス（production）を図９に示す．

実際に行われたゲーミングのデータをシミュレーションの入力として与えることによって，事後分析が可能となる．例えば，特定のチームの入力をマシンエージェントによって代行させ，

```
sub sim_process {

    # 期首初期状態
    carry_forward qw(在庫数 受注残 注文数 配送遅れ); # 前期から引継ぎ
    tlet {
        @{ ${配送遅れ()} } = @{ ${配送遅れ(1)} }; # 配列のコピーは注意が必要
        ${出庫数()} = '-' ;
        ${顧客需要()} =《需要決定処理》;
        ${受注数()} = TEAM==1 ? ${顧客需要()} : ${受注数(1)};
    };
    show_state("[初期状態]");

    # 入庫
    tlet {
        ${在庫数()} += ${配送遅れ()}->[0];
        shift @{${配送遅れ()}};
        push  @{${配送遅れ()}}, '-';
    };
    show_state("[入庫後]");

    # 受注
    tlet {
        ${受注残()} += ${受注数()};
        ${受注数()} = '-';
    };
    show_state("[受注後]");

    # 出庫
    tlet {
        ${出庫数()} = min2( ${在庫数()}, ${受注残()} );
        ${在庫数()} -= ${出庫数()};
        ${受注残()} -= ${出庫数()};
        ${配送遅れ()}->[-1]
            = TEAM==MAXT ? '-' : next_team->{tvar}->{'出庫数'}->[0];
    } reverse with {1};         # ループ処理を逆順に行う
    show_state("[出庫後]");

    # 注文票移動
    tlet {
        ${受注数()} = TEAM==1 ? '-' : prev_team->{tvar}->{'注文数'}->[0];
        if (TEAM==MAXT) { ${配送遅れ()}->[-1] = ${注文数()} };
        ${注文数()} = '-';
    } reverse with {1};         # ループ処理を逆順に行う
    show_state("[注文票移動後]");

    # 発注
    tlet {
        ${注文数()} =《発注意思決定処理》;
    };
    show_state("[発注後]");

    # 期末処理
    write_vars($bsim_db);
}
```

図6　ビールゲームのBSimモジュールによる実装（主要部分のみ抜粋）

```
Period: 10
[初期状態]
工場　　受注(40) 在庫(0 68) 遅れ(80 50) 注文(100) 出庫(-)
一次卸　受注(20) 在庫(0 69) 遅れ(20 50) 注文(40) 出庫(-)
二次卸　受注(4) 在庫(26 0) 遅れ(20 10) 注文(15) 出庫(-)
小売　　受注(8) 在庫(0 4) 遅れ(4 6) 注文(4) 出庫(-)
[入庫後]
工場　　受注(40) 在庫(80 68) 遅れ(50 -) 注文(100) 出庫(-)
一次卸　受注(20) 在庫(20 69) 遅れ(50 -) 注文(40) 出庫(-)
二次卸　受注(4) 在庫(46 0) 遅れ(10 -) 注文(15) 出庫(-)
小売　　受注(8) 在庫(4 4) 遅れ(6 -) 注文(4) 出庫(-)
[受注後]
工場　　受注(-) 在庫(80 108) 遅れ(50 -) 注文(100) 出庫(-)
一次卸　受注(-) 在庫(20 89) 遅れ(50 -) 注文(40) 出庫(-)
二次卸　受注(-) 在庫(46 4) 遅れ(10 -) 注文(15) 出庫(-)
小売　　受注(-) 在庫(4 12) 遅れ(6 -) 注文(4) 出庫(-)
[出庫後]
工場　　受注(-) 在庫(0 28) 遅れ(50 -) 注文(100) 出庫(80)
一次卸　受注(-) 在庫(0 69) 遅れ(50 80) 注文(40) 出庫(20)
二次卸　受注(-) 在庫(42 0) 遅れ(10 20) 注文(15) 出庫(4)
小売　　受注(-) 在庫(0 8) 遅れ(6 4) 注文(4) 出庫(4)
[注文票移動後]
工場　　受注(40) 在庫(0 28) 遅れ(50 100) 注文(-) 出庫(80)
一次卸　受注(15) 在庫(0 69) 遅れ(50 80) 注文(-) 出庫(20)
二次卸　受注(4) 在庫(42 0) 遅れ(10 20) 注文(-) 出庫(4)
小売　　受注(-) 在庫(0 8) 遅れ(6 4) 注文(-) 出庫(4)
read_input: round 10
[発注後]
工場　　受注(40) 在庫(0 28) 遅れ(50 100) 注文(10) 出庫(80)
一次卸　受注(15) 在庫(0 69) 遅れ(50 80) 注文(50) 出庫(20)
二次卸　受注(4) 在庫(42 0) 遅れ(10 20) 注文(10) 出庫(4)
小売　　受注(-) 在庫(0 8) 遅れ(6 4) 注文(10) 出庫(4)
```

図7　ビールゲームシミュレーション実行例

特定の戦略を用いた場合にはどのような結果がもたらされるかを分析する代理シミュレーション（田名部，2011a; 田名部，2011b）を容易に実現することができる．図10は，11チームで実施されたベーカリーゲームの実データを用い，チーム１の意思決定値を変化させたときの剰余金の変化を示したものである．販売価格を500円から1000円まで変化させ，材料調達個数と製造指図数を同じにしながら50から300まで変化させたときの剰余金の変化を，販売価格毎にプロットしている．

横軸は，製造指図数（＝材料調達個数），縦軸は剰余金を示す．図中のマーカー部分１点を計算するのに，10期分のシミュレーションが１回必要となる．このシミュレーションは，機械的にチーム１の意思決定を書き換えて行った，単純代理シミュレーションである．

```
sub sim_process {
    # load input variables
    load_input();

    # business process
    procurement();              # procurement
    production();               # production
    market();                   # market
    sales();                    # sales

    # accounting
    cash_flow();
    income_statement();
    balance_sheet();

    # statistics
    statistics();

    # display
    write_vars($bsim_db);
    out_values();

}
```

図8　ベーカリーゲームのメインプロセス（抜粋）

```
sub production {
    tlet {
        ${生産待ち数()} = min2( ${仕入後材料在庫数()}, ${製造指示()} );
        ${期末材料在庫数()} = ${仕入後材料在庫数()} - ${生産待ち数()};
        ${生産個数()} = ${生産待ち数(1)};
        ${販売可能数()} = ${生産個数()};
        ${累積生産個数()} = ${累積生産個数(1)} + ${生産個数()};
    };
}
```

図9　ベーカリーゲームの生産プロセス

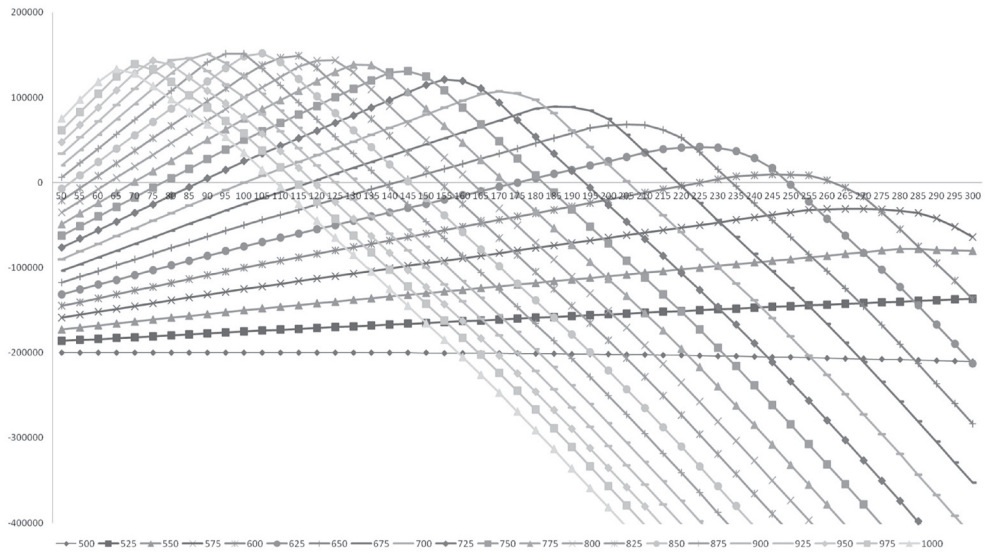

図10　BSimモジュールで実装したベーカリーゲームにおける代理シミュレーション

9．おわりに

本論では，ビジネスゲームを支援するためのウェブサービスに求められるビジネスゲームの実行系のみたすべき機能を特定し，アーキテクチャを示すとともに，その実現のためにYBGやBMDSで用いられているゲーム記述言語BMDLの仕様を基盤とするゲーム記述言語の設計と実装を行い，その利用例を示した．まず，複数の意思決定主体が同じ立場で，特定の市場でビジネスを行うというYBGやBMDSで前提としている典型的なビジネスゲームをオートマトンとして定式化した．システムへの入力は，ゲームプレイヤの意思決定の他に，進行役の制御や，外部からの外乱などがある．また，システムの状態は，各チームの状態と市場の状態の組合せとして表現される．これに基づいて，ビジネスゲームの実行系を，プレイヤのプロセスと市場のプロセスという部分システムからなる入出力システムとして表現し，さらに，この表現を拡張させて，ビジネスゲーム実行系がクラウド上に実装される際のアーキテクチャを提示した．続いて，ビジネスゲーム実行系の核となる言語処理系の設計に必要となるゲームモデルにおける変数の取り扱いについて，BMDLのモデル変数の取り扱いを踏まえて考察し，モデル変数間の関係性を計算処理するための仕組みとして，デリファレンスを用いてDSLを設計するという方法を提案した．そして，この方法に基づく処理系の実装の例としてPerlによるBSimモジュールを示した. 最後に, このBSimモジュールを用いたビールゲームとベーカリーゲームのシミュレータの実装とその利用例を述べた．本論で示したDSLによるシミュレーション実行系は，Perl以外のPython，Ruby，PHP, C#, JavaScriptなどの言語でも実装可能である．今後は，本論で述べた実行系をクラウド環境に実装し，他のウェブサービスと連携させ，より高度なゲーミングシミュレーションの環境構築を行い，その有効性を評価したい．

謝　辞

本研究は，科研費（23530430）ならびに科研費（23330125）の助成を受けたものである．

注　記

1）デスクトップPCやノートPCなど，ゲーム開発者の利用環境上にウェブサーバ，Perl処理系，BMDSをインストールして，ゲーム開発をすることも可能である．

2）この頃より，従来システムと新システムとを区別するために，それまでのシステムはYBG1.0と呼ばれるようになる．

3）BMDLに基づく，ジェネレータgg7ではシリーズ変数は実装されていないが，YBGでは，2003年から実装されている．

4）実際は，Cコンパイラが処理できるように，「売上高」「販売価格」「販売数量」などの日本語部分は，実際はEUCコードに対応するASCII文字に変換される．本論では，分かりやすさのために日本語のまま示してある．

5）Perlでは，文字列を添え字とする配列のことをハッシュ（連想配列）と呼ぶ．添え字は「キー」と呼ばれ，キーで特定される配列要素を「値」と呼ぶ．

参　考　文　献

久野靖（2001）「経営シミュレーションゲーム生成系」，鈴木久敏「高度職業人養成のためのビジネス教育ツールの開発」，科学研究費補助金基盤研究（B）（2）研究成果報告書，pp.90-101.

白井宏明，藤森洋志，久野靖，鈴木久敏，寺野隆雄，津田和彦（2000）「WWW環境を利用したビジネスゲーム開発ツール」，教育システム情報学会誌，Vol. 17，No. 3，pp. 339-348，2000.

白井宏明（2008）「ビジネスゲームを主体とした授業構成に関する考察」，横浜経営研究，Vol.29，No.3，pp.171-188.

田名部元成（2011a）「製造販売型ビジネスゲームにおける需要分配に関する考察」，横浜経営研究，Vol.32，No.1，pp.145-169.

田名部元成（2011b）「代理データシミュレーション手法」，日本シミュレーション＆ゲーミング学会全国大会論文報告集（2011年秋），pp. 21-24.

Tsuda, K., Terano, T., Kuno, Y., Shirai, H., Suzuki, H.（2002）“A compiler for business simulations: Toward business model development by yourselves,” Information Sciences, Vol. 143, pp.99-114.

Sterman, J. D.（1992）“Teaching Takes Off: Flight Simulators for Management Education,” OR/MS Today, pp.40-44.

〔たなぶ　もとなり　横浜国立大学大学院国際社会科学研究科准教授〕

〔2012年１月30日受理〕

付録１　BSimAgent.pm

```
package BSimAgent;
use strict;
use warnings;
use utf8;

sub new {
    my $class = shift;
    bless( {}, $class )->init(@_);
}

sub init {
    my ( $this, %args ) = @_;
    my %attr = (
        id     => undef,
        next   => undef,
        prev   => undef,
        parent => $this
    );    # public attributes

    map { $this->{$_} = $attr{$_} } keys %attr;
    map {
        $this->{$_} = $args{$_} if defined $args{$_}
    } keys %args;
    return $this;
}

sub set {
    my ( $this, %args ) = @_;
    map { $this->{$_} = $args{$_} } keys %args;
    return $this;
}

sub get {
    my ( $this, $key ) = @_;
    return $this->{$key};
}

sub dump {
    my $this = shift;
    print "id = ", $this->{id}, "¥n";
    for my $key ( sort keys %{ $this->{tvar} } ) {
        my $val = $this->{tvar}->{$key};
        print "$key = ";
        print defined $val
          ? ( ref($val) eq 'ARRAY' ? "(@{$val})" : "$val" )
          : 'undef';
        print "¥n";
    }
    return $this;
}
1;
```

付録２ BSimLang.pm

```
package BSimLang;
use strict;
use warnings;
use Agent;
use Data::Dumper;
use utf8;

# exporting functions
use base qw(Exporter);
our @EXPORT = qw(
  def gcon scon tvar tcon tvars ivar
  tlet carry_forward
  MAXT MAXR TEAM ROUND team_id OFFSET
  next_team prev_team
  next_round set_round
  with subset_of where value_of getv outv
  min2 max2 rint pow
  sum_of inverse_sum_of rank_of
  show_vars write_vars read_vars
  clear_round_data );

# simulation general variables
my @agents       = ();
my $max_round    = 1;
my $current_round = 1;
my $max_offset   = 4;

my $gcon_ref_of = {};   # hash ref of refs for the gcon
my $scon_ref_of = {};   # hash ref of refs for the scon
my $tcon_ref_of = {};   # hash ref of refs for the tcon
my $tvar_ref_of = {};   # hash ref of refs for the tvar
my $ivar_ref_of = {};   # hash ref of refs for the ivar

my $sim_name = 'sim';   # default simulation name

#
# YBG metavariable subroutines
#
sub MAXT()     { return @agents - 0 }
sub MAXR()     { return $max_round }
sub TEAM()     { return $_->{id} + 1 }
sub ROUND()    { return $current_round }
sub team_id()  { return $_->{id} }
sub next_team() { return $_->{next} }
sub prev_team() { return $_->{prev} }
sub OFFSET()   { return $max_offset }

#
# YBG simulation control subroutines
#
sub next_round {
    for my $obj (@agents) {
        for my $key ( keys %$tvar_ref_of ) {
            unshift @{ $obj->{tvar}->{$key} }, undef;
            pop @{ $obj->{tvar}->{$key} };
        }
        for my $key ( keys %$ivar_ref_of ) {
            unshift @{ $obj->{ivar}->{$key} }, undef;
            pop @{ $obj->{ivar}->{$key} };
        }
    }
    $current_round++;
}

sub set_round {
    my ( $spec_round ) = shift;

    if ( defined $spec_round ) {
        $spec_round <= MAXR || die "specified round is larger
than MAXR";
        $current_round = $spec_round;
        return $spec_round;
    }
    else {
        return undef;
    }
}

#
# YBG game setting commands
#
sub def {
    my ( $op, $val ) = @_;
    defined $op || die "def def_command parameter¥n";

    if ( $op eq 'maxteam' ) {
        defined $val or
          die "usage: def maxteam <number_of_teams>¥n";
        $val > 0 or
          die "number of teams must be positive integer.¥n";

        for my $i ( 0 .. $val - 1 ) {
            push @agents,
                BSimAgent->new( id => $i, agent => ¥@agents );
```

```
        }

        # make link 'next'
        for my $i ( 0 .. $val - 2 ) {
            $agents[$i]->{next} = $agents[ $i + 1 ];
        }

        # make link 'prev'
        for my $i ( 1 .. $val - 1 ) {
            $agents[$i]->{prev} = $agents[ $i - 1 ];
        }
    }
    elsif ( $op eq 'maxround' ) {
        defined $val or
          die "usage: def maxround max_number_of_rounds¥n";
        $val > 0 or
          die "number of rounds must be positive integer.¥n";
        $max_round = $val;
    }
    elsif ( $op eq 'simname' ) {
        defined $val or
          die "usage: def simname name_of_simulation¥n";
        $sim_name = $val;
    }
}

#
# YBG variable definition commands
#
sub gcon {
    my ( $gcon_name, $gcon_value ) = @_;

    $gcon_ref_of->{$gcon_name} = ¥$gcon_value;

    no strict 'refs';
    ${ 'main::' . $gcon_name } = $gcon_ref_of->{$gcon_name};
    *{ 'main::' . $gcon_name }
      = sub { return $gcon_ref_of->{$gcon_name} };
}

sub scon {
    my ( $scon_name, @scon_values ) = @_;

    $scon_ref_of->{$scon_name} = undef;

    # scon entry into (symbol=>ref) hash table
    @{ $scon_ref_of->{$scon_name} }
      = _expand(@scon_values);

    no strict 'refs';

    # create function whose name is same as scon
    *{ 'main::' . $scon_name } = sub {
        my $round_offset = shift;
        if ( !defined $round_offset ) { $round_offset = 0 }
        return ¥$scon_ref_of->{$scon_name}
            ->[ $current_round - $round_offset ];
      }
}

sub tcon {
    my ( $tcon_name, @tcon_values ) = @_;
    $tcon_ref_of->{$tcon_name} = undef;

    # tcon entry into (symbol=>ref) hash table
    @{ $tcon_ref_of->{$tcon_name} }
      = _expand(@tcon_values);

    # value entry for each agent
    for my $agent (@agents) {
        my $val
            = $tcon_ref_of->{$tcon_name}->[ $agent->{id} ];
        $agent->{tcon}->{$tcon_name}
            = defined $val ? $val : 0;
    }

    # create function whose name is same as tcon_name
    no strict 'refs';
    *{ 'main::' . $tcon_name }
        = sub { return ${ 'main::' . $tcon_name } };
}

sub tvar {
    my ( $tvar_name, @init_tvar_values ) = @_;

    # tvar entry into (symbol=>ref) hash table
    $tvar_ref_of->{$tvar_name} = undef;

    # value entry for each agent
    for my $agent (@agents) {
        @{ $agent->{tvar}->{$tvar_name} } =
          ( undef, _expand(@init_tvar_values) );
    }

    # create function whose name is same as tvar_name
    no strict 'refs';
    *{ 'main::' . $tvar_name } = sub {
        my $round_offset = shift;
        $round_offset = 0 if !defined $round_offset;
        return
            ${ 'main::' . $tvar_name . '@' . $round_offset };
    };
}

sub tvars {
    my ( @tvars ) = @_;
     map { tvar $_ } @tvars;
}

sub ivar {
    my ( $ivar_name, %ivar_args ) = @_;

    # tvar entry into (symbol=>ref) hash table
    $ivar_ref_of->{$ivar_name} = undef;

    # value entry for each agent
    for my $agent (@agents) {
        @{ $agent->{ivar}->{$ivar_name} } = ( undef );
        %{ $agent->{iarg}->{$ivar_name} } = %ivar_args;
    }

    # create function whose name is same as tvar_name
    no strict 'refs';
    *{ 'main::' . $ivar_name } = sub {
        my $round_offset = shift;
        $round_offset = 0 unless defined $round_offset;
        return
            ${ 'main::' . $ivar_name . '@' . $round_offset };
    };
}

#
# YBG model calculation commands
#
sub tlet(&@) {
    my ( $code, @agentset ) = @_;
    @agentset = @agents if !@agentset;

    for my $obj (@agentset) {
        _set_param($obj);
        $_ = $obj;
        $code->();
    }
}

sub carry_forward {
    my @tvars = @_;

    for my $tvar (@tvars) {
        no strict 'refs';
        tlet { ${ &{ 'main::' . $tvar }() }
           = ${ &{ 'main::' . $tvar }(1) } };
    }
}

sub getv(&$) {
    my ( $code, $team ) = @_;

    my $obj = $agents[$team -1];
    _set_param($obj);
    $_ = $obj;
    $code->();
}
```

```
sub outv(&$;$) {
    my ( $code, $label, $fmt ) = @_;;
    $fmt = "%d" unless defined $fmt;

    print $label;
    for my $obj (@agents) {
        _set_param($obj);
        $_ = $obj;
        printf ",$fmt", $code->();
    }
    print "¥n";
}


#
# Agentset commands
#
sub with(&;@) {
    my ( $condition, @agentset ) = @_;
    @agentset = @agents if !@agentset;
    my @ret;

    for (@agentset) {
        push @ret, $_ if $condition->();
    }
    return @ret;
}

sub subset_of(&@) {
    my $array_code = shift;
    my $condition  = shift;
    my @ret;

    for ( $array_code->() ) {
        push @ret, $_ if $condition->();
    }
    return ( @ret, @_ );
}

sub where(&) {
    my $code = shift;
    return $code;
}

sub value_of {
    my $val = shift;
    return defined $val ? $val : 'undef';
}

sub show_vars {
    tlet {
        my $obj = $_;
        print "id: ", $obj->{id}, "¥n";

        print "ivar¥n";
        for my $ivar ( keys %{ $obj->{ivar} } ) {
            print $ivar, "¥n";
            print Dumper( $obj->{ivar}->{$ivar} );
            my $val_ref = $obj->{ivar}->{$ivar};
            print "$ivar = ( ";
            if ( @{$val_ref} > 0 ) {
                for my $val ( @{$val_ref} ) {
                    print $val . " " if defined $val;
                }
            }
            print ");¥n";
        }

        print "tvar¥n";
        for my $tvar ( keys %{ $obj->{tvar} } ) {
            print $tvar, "¥n";
            print Dumper( $obj->{tvar}->{$tvar} );
            my $val_ref = $obj->{tvar}->{$tvar};
            print "$tvar = ( ";
            if ( @{$val_ref} > 0 ) {
                for my $val ( @{$val_ref} ) {
                    print $val . " " if defined $val;
                }
            }
            print ");¥n";
        }
    };
}

sub write_vars {
    my ( $db, $offset ) = @_;
    $offset = 0 unless defined $offset;

    my $vars = {};
    tlet {
        my $obj = $_;
        my $team = $obj->{id} + 1;
        for my $ivar ( keys %{ $obj->{ivar} } ) {
            $vars->{$team}->{$ivar}
              = $obj->{ivar}->{$ivar}->[$offset];
        }
        for my $tvar ( keys %{ $obj->{tvar} } ) {
            $vars->{$team}->{$tvar}
              = $obj->{tvar}->{$tvar}->[$offset];
        }
    };
    $db->write_round_data( ROUND - $offset, $vars );
}

sub read_vars {
    my ( $db ) = @_;
    for (my $i=1; $i<=$max_offset; $i++) {
        print "processing offset $i¥n";

        my $VAR1 = $db->read_round_data( ROUND - $i );

        if ( !defined $VAR1 ) {
            print "Reading data of round ",
                (ROUND - $i), "... ";
            print "Error! file r",
                (ROUND-$i), ".txt not found.¥n";
            die;
        }

        tlet {
            my $obj = $_;
            my $team = $obj->{id} + 1;
            for my $ivar ( keys %{ $obj->{ivar} } ) {
                $obj->{ivar}->{$ivar}->[$i]
                    = ROUND > $i ?
                      $VAR1->{$team}->{$ivar} : undef;
            }
            for my $tvar ( keys %{ $obj->{tvar} } ) {
                $obj->{tvar}->{$tvar}->[$i]
                    = ROUND > $i ?
                      $VAR1->{$team}->{$tvar} : undef;
            }
        };
    }
}

sub clear_round_data {
    my ( $db, $round ) = @_;

    if ( defined $round ) {
        print "removing data of round ", $round, " ... ";
        my $n = $db->remove_round_data( $round );
        if ( defined $n ) {
            if ($n ==1 ) {
                print "deleted¥n";
                die;
            }
            else {
                print "error occured when removing files¥n";
                die;
            }
        }
        else {
            print "file not found¥n";
            die;
        }
    }
}

#
# YBG Math Library
#
sub min2 {
    my ( $a, $b ) = @_;
```

```
    return $a < $b ? $a : $b;
}

sub max2 {
    my ( $a, $b ) = @_;
    return $a > $b ? $a : $b;
}

sub rint {
    my $val = shift;
    return int( $val + 0.5 );
}

sub pow {
    my ( $a, $b ) = @_;
    return $a**$b;
}

sub sum_of(&;@) {
    my ( $code, @agentset ) = @_;
    my $sum = 0;
    @agentset = @agents if !@agentset;

    for my $obj (@agentset) {
        _set_param($obj);
        $sum += $code->();
    }
    return $sum;
}

sub inverse_sum_of(&;@) {
    my ( $code, @agentset ) = @_;
    my $sum = undef;
    @agentset = @agents if !@agentset;

    for my $obj (@agentset) {
        _set_param($obj);

        if ( !defined $sum ) {
            $sum = $code->();
        }
        else {
            my $val = $code->();
            if ( $val != 0 ) {
                $sum = ( $sum * $val ) / ( $sum + $val );
            }
        }
    }
    return $sum;
}

sub rank_of(&;@) {
    my ($code, @agentset) = @_;
    @agentset = @agents if !@agentset;

    my (@values,@rank);

    for my $obj (@agentset){
              _set_param($obj);
              push @values, $code->();
    }

    for my $x (@values){
              my $r = 1;
              for my $y (@values) { $r++ if $y>$x }
              push @rank, $r;
    }
    return @rank;
}

#
# internal subroutines
#
sub _expand {
    my @array = @_;
    my @ret   = ();

    for my $obj (@array) {
        if ( ref $obj eq 'ARRAY' ) {
            push @ret, [ _expand( @{$obj} ) ];
        }
        elsif ( ref $obj eq 'HASH' ) {
            push @ret, { _expand( %{$obj} ) };
        }
        else {
            push @ret, ( defined $obj ? $obj : 0 );
        }
    }
    return @ret;
}

sub _set_param {
    my $obj = shift;

    # scon
    for my $var ( keys %$scon_ref_of ) {
        no strict 'refs';
        # default
        ${ 'main::' . $var }
          = ¥$scon_ref_of->{$var}->[$current_round];

        # ref to array of values
        ${ 'main::' . $var . '@' } = $scon_ref_of->{$var};

        for my $i ( 0 .. $max_offset ) {
            ${ 'main::' . $var . '@' . $i }
              = ¥$scon_ref_of->{$var}
                ->[ $current_round + $i ];
        }
        use strict 'refs';
    }

    # tcon
    for my $var ( keys %$tcon_ref_of ) {
        no strict 'refs';
        # default
        ${ 'main::' . $var } = ¥$obj->{tcon}->{$var};
        use strict 'refs';
    }

    # tvar
    for my $var ( keys %$tvar_ref_of ) {
        no strict 'refs';
        # default
        ${ 'main::' . $var } = ¥$obj->{tvar}->{$var}->[0];

        # ref to array of values
        ${ 'main::' . $var . '@' } = $obj->{tvar}->{$var};

        for my $i ( 0 .. $max_offset ) {
            if ( !defined $obj->{tvar}->{$var}->[$i] ) {
                $obj->{tvar}->{$var}->[$i] = 0;
            }

            ${ 'main::' . $var . '@' . $i }
              = ¥$obj->{tvar}->{$var}->[$i];
        }
        use strict 'refs';
    }

    # ivar
    for my $var ( keys %$ivar_ref_of ) {
        no strict 'refs';
        # default
        ${ 'main::' . $var } = ¥$obj->{ivar}->{$var}->[0];

        # ref to array of values
        ${ 'main::' . $var . '@' } = $obj->{ivar}->{$var};

        for my $i ( 0 .. $max_offset ) {
            if ( !defined $obj->{ivar}->{$var}->[$i] ) {
                $obj->{ivar}->{$var}->[$i] = 0;
            }

            ${ 'main::' . $var . '@' . $i }
              = ¥$obj->{ivar}->{$var}->[$i];
        }
        use strict 'refs';
    }
}

1;
```

付録３　BSimData.pm

```
package BSimData;
use FileHandle;
use strict;
use warnings;
use Data::Dumper;

# exporting functions
use base qw( Exporter );
our @EXPORT = qw(
 read_input write_round_data
 read_round_data remove_round_data);

sub new {
    my $class = shift;
    bless( {}, $class )->init(@_);
}

sub init {
    my ( $this, %args ) = @_;
    my %attr = (
        data_dir => './DATA',
        parent => $this
    );    # public attributes

    map { $this->{$_} = $attr{$_} } keys %attr;
    map {
       $this->{$_} = $args{$_} if defined $args{$_}
    } keys %args;

    mkdir $this->{data_dir},
          0755 unless -d $this->{data_dir};
    return $this;
}

sub set {
    my ( $this, %args ) = @_;
    map { $this->{$_} = $args{$_} } keys %args;
    return $this;
}

sub get {
    my ( $this, $key ) = @_;
    return $this->{$key};
}

sub dump {
    my $this = shift;
    print "id = ", $this->{id}, "¥n";
    for my $key ( sort keys %{ $this->{tvar} } ) {
        my $val = $this->{tvar}->{$key};
        print "$key = ";
        print defined $val
          ? ( ref($val) eq 'ARRAY' ? "(@{$val})" : "$val" )
          : 'undef';
        print "¥n";
    }
    return $this;
}

#
# Read input data
#
sub read_input {
    my ( $this, $round ) = @_;            # round
    die "read_input_data: round not defined"
        unless defined $round;

    my $dir = $this->{data_dir} . "/input";
    return undef unless -e $dir . "/r" . $round . ".txt";

    print "read_input: round $round¥n";
    open my $fh, "<", $dir . "/r" . $round . ".txt"
        || die "can not open file $!¥n";
    my @inputs = <$fh>;
    close $fh;

    my $VAR1;
    eval( join q{}, @inputs );

    my $ret = {};
    for my $record (@{$VAR1}) {
        $ret->{$record->{team}} = {};
        my @ivars = grep { $_ ne 'team' } keys %{$record};
        map {$ret->{$record->{team}}->{$_}
               = $record->{$_}} @ivars;
    }
    return $ret;
}

#
# Write round data
#
sub write_round_data {
    my ( $this, $round, $data_ref ) = @_;
    die "write_round_data: round not defined" unless defined
$round;

    my $dir = $this->{data_dir} . "/output";
    open my $fh, ">", $dir . "/r" . $round . ".txt"
        || die "can not open file $!¥n";
    print $fh Dumper( $data_ref );
    close $fh;
}

#
# Read round data
#
sub read_round_data {
    my ( $this, $round ) = @_;
    die "read_round_data: round not defined" unless defined
$round;

    my $dir = $this->{data_dir} . "/output";
    mkdir $dir, 0755 unless -d $dir;

    return undef unless -e $dir . "/r" . $round . ".txt";

    print "read_round_data: round $round¥n";
    open my $fh, "<", $dir . "/r" . $round . ".txt"
        || die "can not open file $!¥n";
    my @inputs = <$fh>;
    close $fh;

    my $VAR1;
    eval( join q{}, @inputs );
    return $VAR1;
}

#
# Remove round data
#
sub remove_round_data {
    my ( $this, $round ) = @_;
    die "remove_round_data: round not defined"
        unless defined $round;

    my $dir = $this->{data_dir} . "/output";
    my $file = $dir . "/r" . $round . ".txt";
    if ( -e $file ) {
        return unlink $file;
    }
    else {
        return undef;
    }
}

1;
```

付録４　ビールゲームシミュレータ

本論で紹介したビールゲームのシミュレーションは，付録１～３に挙げたBSimモジュールと，beer.plおよびconf.plによって実施することができる．それぞれのファイルは図11に示すように配置される．本シミュレータは，Windows, Mac OS X, LinuxのPerl 5.10以降が動作する環境であれば，いずれでも動作する．シミュレーションの本体は，beer.plである．このファイルを実行することでシミュレーションを行うことができる．

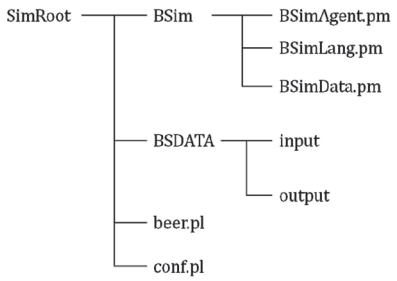

図11　シミュレーションファイル

なお，inputディレクトリに，チームの入力データを事前に保存しなければならない．各期のすべてのチームの入力は，次の形式で，r(期番号).txt (r1.txt, r2.txt, …)というファイル名で保存される必要がある．

```
$VAR1 = [
        {
          "¥x{6ce8}¥x{6587}¥x{6570}" => '4',
          'team' => 1
        },
        {
          "¥x{6ce8}¥x{6587}¥x{6570}" => '4',
          'team' => 2
        },
        {
          "¥x{6ce8}¥x{6587}¥x{6570}" => '4',
          'team' => 3
        },
        {
          "¥x{6ce8}¥x{6587}¥x{6570}" => '4',
          'team' => 4
        }
      ];
```

ここで，\x{6ce8}\x{6587}\x{6570}は，「注文数」をUTF8で表現したものである．この例では，team 1 の 注文数は4, team 2の注文数は4, … を表す．

beer.pl

```
#!/usr/bin/perl -w
#
# beer.pl - YBG like Agent Based Simulator for Beer Game
#
# Copyright (C) Motonari Tanabu <tanabu@ynu.ac.jp>
# Yokohama National University
# November 14, 2010 - December 31, 2011.
#
BEGIN { push @INC, './BSim' }    # Library Path
use strict qw(refs vars);
use warnings;
use utf8;
use Config;
use BSimLang;
use BSimData;
use Data::Dumper;
require "conf.pl";                    # read simulation
parameter setting

# Simulation Parameters
my  $sim_params  =  get_param();        #  get  simulation
parameters in ARGV
map {
    print $_, " = ",
          value_of( $sim_params->{$_}->{val} ), "¥n"
} sort keys %{$sim_params};
print "¥n";

# Simulation Data
my $data_dir = get_data_dir(); # data directory
my $bsim_db = BSimData->new( data_dir => $data_dir ); # db

# Specifiled round
my $spec_round = $sim_params->{round}->{val};

# Cleaning Data
clear_round_data($bsim_db,
  $sim_params->{clear_round}->{val} );

#
# Simulation Model
#
def simname  => "Beer Game";
def maxteam  => 4;
def maxround => 34;

# series constant
scon 顧客需要;

# global constant

# team constant
tcon 段階名=>qw(小売 二次卸 一次卸 工場);
tcon 標準在庫数=>12, 12, 12, 12;
#tcon 標準発注数=>12, 16, 20, 24;

# input variable
tvar 注文数 =>4;

# team variable
tvar 受注数 =>  4;
tvar 在庫数 => 12;
tvar 受注残 =>  0;
tvar 配送遅れ => [4, 4];    # same as The Beer Game
tvar 出庫数;

#
# Simulation Control
#
set_round( $spec_round );
if ( ROUND == 1 ) {
    map { write_vars $bsim_db, $_ } ( 1.. OFFSET );
}
else {
    read_vars( $bsim_db );
}

if ( defined $spec_round ) {    # Execution round specified
    sim_process();
}
else {                      # All rounds execution
    for ( ROUND .. MAXR ) {
        sim_process();
        next_round();
    }
}

#
# Simulation Process Model
#
sub sim_process {
    print "Period: ", ROUND, "¥n";

    # 期首初期状態
    carry_forward qw(在庫数 受注残 注文数 配送遅れ);
    tlet {
        @{ ${配送遅れ()} } = @{ ${配送遅れ(1)} };
        ${出庫数()} = '-' ;
        ${顧客需要()} = 《省略》;
        ${受注数()} = TEAM==1 ? ${顧客需要()} : ${受注数(1)};
    };
    show_state("[初期状態]");

    # 入庫
    tlet {
        ${在庫数()} += ${配送遅れ()}->[0];
        shift @{${配送遅れ()}};
        push  @{${配送遅れ()}}, '-';
    };
    show_state("[入庫後]");

    # 受注
    tlet {
        ${受注残()} += ${受注数()};
        ${受注数()} = '-';
    };
    show_state("[受注後]");

    # 出庫
    tlet {
        ${出庫数()} = min2( ${在庫数()}, ${受注残()} );
        ${在庫数()} -= ${出庫数()};
        ${受注残()} -= ${出庫数()};
        ${配送遅れ()}->[-1]
             = TEAM==MAXT ?
               '-' : next_team->{tvar}->{'出庫数'}->[0];
    } reverse with {1};       # ループ処理を逆順に行う
    show_state("[出庫後]");

    # 注文票移動
    tlet {
        ${受注数()}
            = TEAM==1 ?
              '-' : prev_team->{tvar}->{'注文数'}->[0];
        if (TEAM==MAXT) { ${配送遅れ()}->[-1] = ${注文数()} };
        ${注文数()} = '-';
    } reverse with {1};
    show_state("[注文票移動後]");

    # 発注
    my $input = $bsim_db->read_input( ROUND );
    if ( defined $input ) {
        tlet {
            ${注文数()} = $input->{ TEAM() }->{'注文数'};
        };
    }
    else {
        print "Warning! input data not defined¥n";
        tlet {
            ${注文数()}
              = max2(0, ${標準在庫数()} - ${在庫数()});
        };
    }
    show_state("[発注後]");

    # 期末処理
    write_vars($bsim_db);
}
```

```
#
# simulation specific subroutines
#
sub show_state {
    my @messages = @_;
    print @messages, "¥n";
    tlet {
        print ${段階名()}, "¥t";
        print " 受注(", ${受注数()}, ")";
        print " 在庫(", ${在庫数()}, q{ }, ${受注残()}, ")";
        print " 遅れ(", join(q{ }, @{${配送遅れ()}}), ")";
        print " 注文(", ${注文数()}, ")";
        print " 出庫(", ${出庫数()}, ")";
        print "¥n";
    } reverse with {1};
}
__END__
```

conf.pl

```
#
# conf.pl - Simulation configuration
#

# Data Directory
my $data_dir = 'BSDATA';

# Sim Parameters
my $sim_params = {
    round      => { opt => '-r',    val => undef },
    clear_round => { opt => '-cr',   val => undef },
};

# Character Set
binmode STDOUT => $Config{'osname'} eq 'darwin'
    ? ":utf8"
    : ":encoding(shift_jis)";

# get data dir
sub get_data_dir { return $data_dir }

# get simparam
sub get_param {
    for ( my $i = 0; $i < @ARGV; ) {
        my $getarg = sub { return $ARGV[ $i++ ] };
        my $argv = $getarg->();

        map {
            my $var = $sim_params->{$_};
            if ( $argv eq $var->{opt} ) {
                my $v = $getarg->();
                my ( $max, $min )
                    = ( $var->{min}, $var->{max} );
                if ( range( $v, $min, $max ) ) {
                    $var->{val} = $v
                }
                else { die $var->{opt} . " <" . $_ . ">¥n" }
            }
        } keys %{$sim_params};

        # local subroutine
        sub range {
            my ( $x, $a, $b ) = @_;
            if ( defined $x && $x =~ m/^¥d+$/ ) {
                if ( defined $a && defined $b ) {
                    return $x >= $a && $x <= $b;
                }
                elsif ( defined $a && !defined $b ) {
                    return $x >= $a;
                }
                elsif ( !defined $a && defined $b ) {
                    return $x <= $b;
                }
                else {
                    return 1;
                }
            }
            else {
                return 0;
            }
        }
    }
    return $sim_params;
}

1;
```