XGMTK report No.6
2013 年 12 月

Lore

TRPG ルール、シナリオ記述向け言語

概説

椎路 ちひろ

筆者はテーブルトーク・ロール・プレイング・ゲーム（TRPG）の様々なシステムのルール及びシナリオを記述するための言語として Lore を開発しています。Lore 言語は、以前 XML ベースの Lore ML として開発していましたが、流石に人間には読み書きしにくいところが多々あったので XML ベースではない普通の（？）言語として、基本的なアイディアはそのままに再設計しました。本稿ではこの文法に基づくサンプル・プログラムを例にとって Lore ML 言語がどのようにして TRPG のルールやシナリオを表現するかについて概説します。

Lore 言語処理系の実装は 2013 年 12 月現在進行中であり、本稿の内容は将来変更される可能性があります。また基本的なアイディアの多くは XGMTK report No.03 で解説した Lore ML 言語から継承していますが、改良のためいくらか変えた部分もあります。

開発中の Lore 言語処理系の最新のコードは以下のように GitHub 上で公開しています：

https://github.com/TakayukiKando/Lore

目次

1 Lore 言語とは

この節では Lore 言語の目標や概要について簡単に述べます。設計にあたっての背景や課題のより詳細な考察は XGMTK report の No.2、No.5 で既に行っていますので興味を持たれた方は参照なさってください。

1.1 オンライン・セッション・システムと記述言語

TRPG（テーブル・トーク・ロール・プレイイング・ゲーム）は元来顔を突き合わせての会話やデータの交換で進行していく、いわゆるテーブル・ゲームの一種です。歴史的には個人レベルの戦闘シミュレーション・ゲームの流れをくんでおり、これに即興演劇の要素が合流したようなシステムになっています。

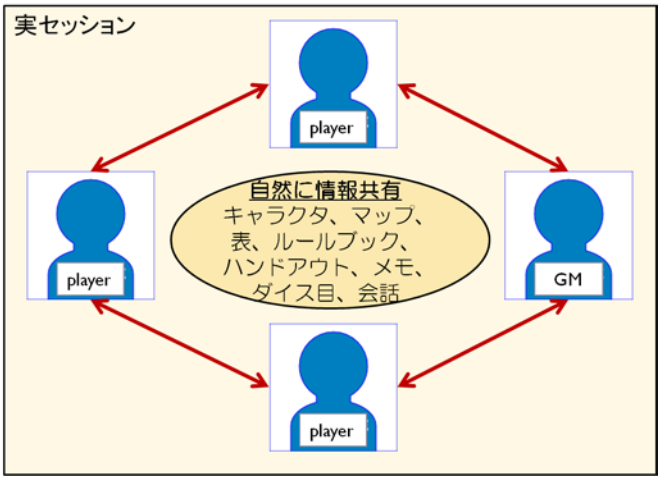

図　1：通常の TRPG セッション

図　1 にも示すように通常の TRPG セッションでは同じ場に会して、レフェリー兼進行役の GM（ゲームマスター）と参加するプレイヤー達が直接の会話と紙に記したテキストや図の手渡し、テーブル上に置いて共有することによって特に意識することなく伝統的かつ自然な方法で様々な情報を交換、共有しています。

オンライン・セッションとは情報機器とネットワークの普及を背景に、互いに離れた GM やプレイヤーたちがこの TRPG のセッションをネットワーク越しに行うことを助けるシステムです。

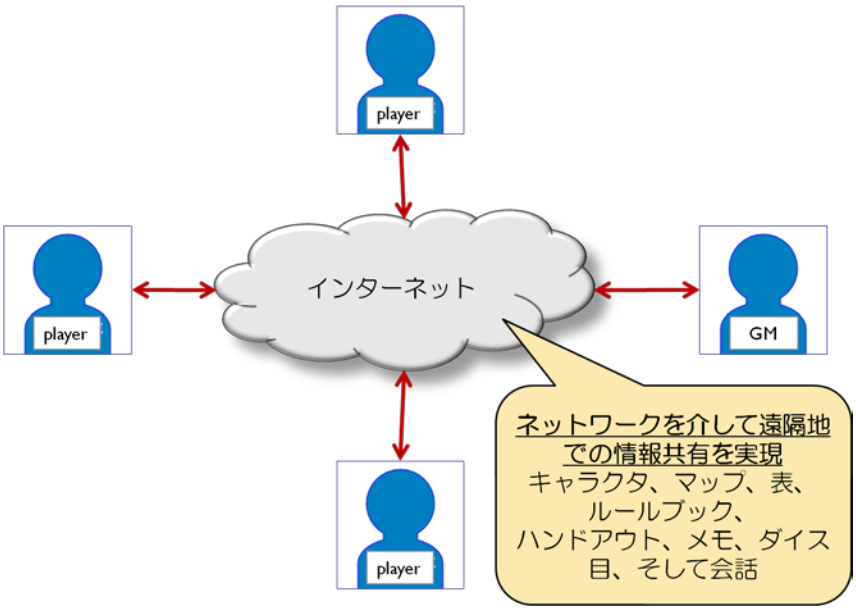

図 2：オンライン・セッション

オンライン・セッションでは図 2に示すように通常の TRPG セッションで交換・共有されていた情報をネットワーク越しに交換、共有できるようにしなければなりません。それを実現するのがオンライン・セッション・システムです。

オンライン・セッション・システムにルール、シナリオ記述言語が必須というわけではありません。チャット・システムを軸に判定表やダイスといった公正さのためにチートできないレフェリング支援システムに対象を限れば汎用的かつ包括的なルール、シナリオ記述システムまでは必要ないでしょう。また HTML ファイルや画像など様々なファイル形式の規格も今や豊富にあり Web ブラウザで広くサポートされており、個々のデータの交換・共有に専用の記述言語は必要ではないでしょう。

そのような理由で現状では汎用のルール、シナリオ記述言語を利用せず、チャット・システムに簡易なコマンド・インタプリタを組み込み、GM やユーザがチャットの合間にテキストでコマンド文字列を入力してシステムの機能を呼び出すコマンド主導型のオンライン・セッション・システムが主流です。

しかしコマンド主導型オンライン・セッション・システムは開発し易いですが、ユーザはコマンドを覚える必要がありますし、コマンドの選択と実行タイミングがユーザに任されるため、ゲームのルールへの理解がばらついている時に正しくセッションを進行するのが難しくなり、それをサポートするための会話がゲームの進行に必須の会話に対して増えることになり、セッションのテンポを悪くする可能性があります。リアルに顔を突き合わせてのセッションより会話で伝達できる情報が減るため、オンライン・セッションでそのリスクは高くなると考えられます。そしてそれはシミュレーション色の強い複雑なルール

の TRPG で顕著になると予想できます。

そこで考えられるのがルール主導型オンライン・セッション・システムです。ルール主導型オンライン・セッション・システムではシステムがゲームのルールを理解し、より積極的にゲームの進行に関与します。ゲーム内の時間や手順の進行を補佐しつつ、状況を読みとって、ユーザからのコマンド入力を待つことなく、状況に合わせた提案メニューをシステム側から提示します[1]。

コマンド主導型とルール主導型を対比したのが図 3です：

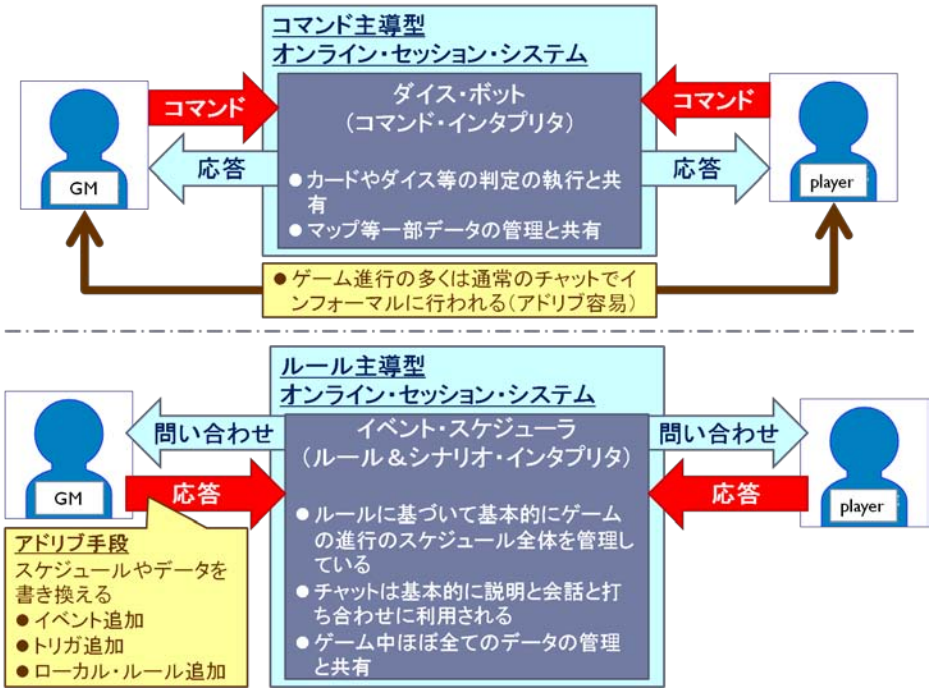

図 3：ルール主導型オンライン・セッション・システム

このようにルール主導型システムはシステムから積極的に GM やプレイヤーへの提案を行うためシステムがルール、ゲームの進行を解釈する必要があります。このため個々のデータ（キャラクタ、マップ、表など）だけでなく、ゲームの進行手順まで含めたルール、そしてシナリオの進行状況といったデータをシステムが処理する形で記述する必要があります。従って様々な TRPG のルール・セットに対応可能な汎用のルール主導型オンライン・セッション・システムを実現するには、プログラムが解釈可能な形式を備えた汎用のルール、シナリオ記述フォーマットが必要となります。

---

[1] システムがルールを理解して GM やプレイヤーに提案するとはいってもルールブックが不要になるわけではありません。ルール主導型オンライン・セッション・システムは進行をスムーズにはしますが、良いシナリオ、良いセッションには依然としてルールの十分な理解が必要だからです。

現在我々が開発している XGMTK オンライン・セッション・システムで利用する記述フォーマットが Lore 言語です。図 4 に示すように Lore 言語を解釈する Lore 言語エンジンはサーバ側でもクライアント側でも中心的な役割を果たすモジュールとなり、セッションの進行とデータの交換・共有を支援します。

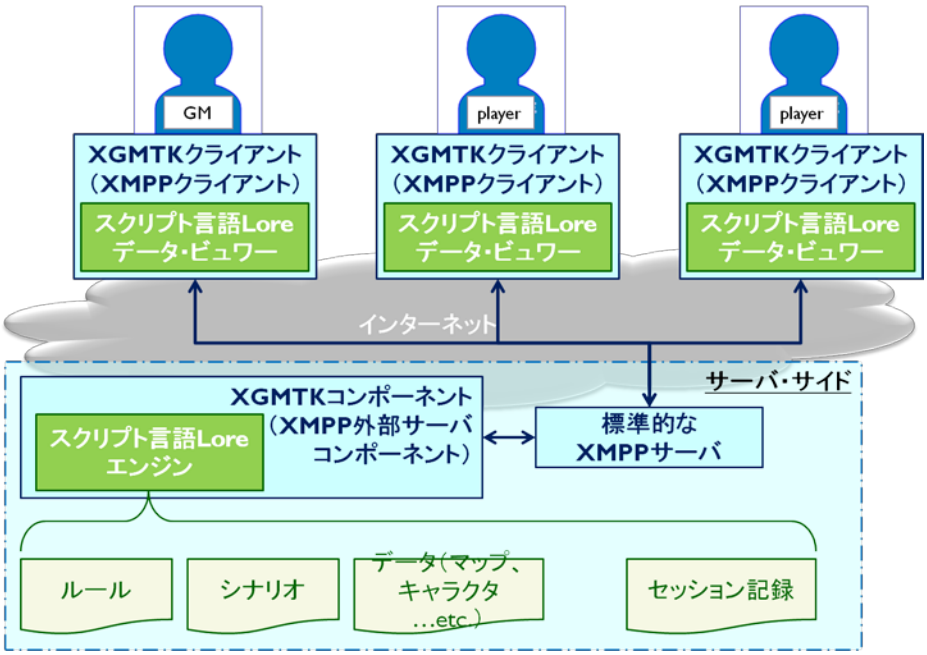

図 4 : XGMTK オンライン・セッション・システム

ただし、ルール主導型オンライン・セッション・システムではルールとシナリオの記述作業が大変になるのでそれを支援するツールであるオーサリング・ツールも必要となります。図 5 に示すように Lore 言語エンジンはオーサリング・ツールにも組み込まれ、専用エディタやテスト環境（セッション・シミュレータ）の実現に利用されます。

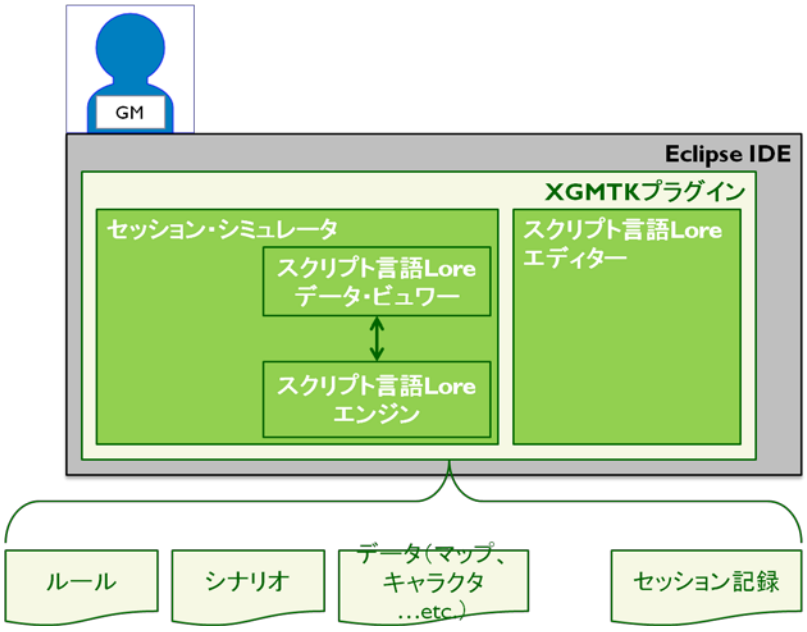

図 5 : XGMTK オーサリング・ツール

1.1 Lore 言語の目標

ルール主導型オンライン・セッション・システムとオーサリング・ツールのために TRPG のルール、シナリオを記述する言語は以下のような特徴をサポートすることが望ましいと筆者は考えています：

(A) 非リアルタイム性

TRPG セッションはプレイヤーにとってリアルタイム（実時間）で進行する必要はありません。プレイヤーの実時間の制約やプレイヤーの身体の反応に左右されないことで凡人でも転載的な判断や英雄的な行為が可能になるという面もあります。

(B) シミュレーション性（時間管理、スケール）

その一方で TRPG はシミュレーション的な側面を持っています。シビアな状況設定を知的に解決するスリル、異世界であってもフィクションとして設定できる情報量には限りがあり、それ以外ではリアルの常識を援用することによる臨場感が持ち味の一つでもあります。その際、ルール化された世界の管理はコンピュータが得意とするところです。

(C) ユーザ・インタラクションのための並列性

(D) 情報密度の偏り（時間的偏り、空間的偏り）

そしてオーサリングの手間削減のため以下のようなことに配慮する必要があります：

(E) 情報密度の偏りを利用した記述圧縮

(F) 部品化による再利用性促進

(G) ユーザ階層を意識した構成

以上に配慮して筆者は以下のようなアプローチを取って TRPG ルールとシナリオの記述言語 Lore を設計しました：

- 離散時間でイベント駆動な実行システム
  - 汎用で厳密なゲーム内時間で管理されたイベント・キューに基づいてセッションが進行
  - 情報密度に応じてイベント発行数をコントロールすることで情報の密度に応じたゲーム進行が可能
  - イベント駆動実行システムに統合されたユーザとの並行インタラクション
- データ構造とルールを記述の分離
  オーサリング・ツールで記述し易い比較的静的なデータ構造とプログラミング作業の必要な動的なデータ構造とルール記述の割合が自然に配分されるので図 6 (11) に示すようにユーザの階層毎に作るものを分けた協働が可能となります。
- TRPG 向けデータ型の充実
  - 範囲型、単位型、ダイス型、ロケーション/エリア型等の導入
- 階層型マップシステムにより情報の密度に応じて位置情報の記述量を選択可能
  - システムにより情報の密度に応じて位置情報の記述量を選択可能
- ロケーション/エリア型の設計上の工夫
  - 複数のファイルから集積してセッションを構成でき再利用性向上
  - タイプの違うマップ表現の混在により情報の密度に応じて位置情報の記述量を選択可能
- ANTLR 4 の採用
  - オーサリング・ツールを作成し易い
    Lore 言語は ANTLR 4 を利用して文法を定義しています。ANTLR ツールにより解析イベント・リスナや解析木ビジタを自動生成し、API として提供するすることでデータの作成を支援するオーサリング・ツールの作成は楽になります。

1.2 Lore 言語ファイルの役割分担

Lore 言語で記述されたファイルは共通の記述形式でありながらも、その用途によって性質が異なり、作成者に求められる知識も異なります。現在表 1に示すような類型を考えています。

表 1 : Lore を構成するファイルと Lore ファイルの例

Lore言語と共に配布されるファイル

ファイル名	作成者	説明
basic.lore	Lore言語開発者	共通する基本的なデータ型の定義

Lore言語で記述されるファイルの例

ファイル名の例	主な作成者	内容の例
RuleAA.lore	ルール設計者、移植者	*ゲームシステムAA*のルール定義(**主に手続き**、単位系、カテゴリや状態を表す名前、etc.)
WorldAA.lore	設定ライタ、移植者	*ゲームシステムAA*の世界設定(**主にデータ**、幾らかの手続き: マップ、クリーチャー、著名NPC、標準的なアイテム設定、ローカル・ルール、etc.)
CampaignBB.lore	キャンペーン(シナリオ)ライタ	*キャンペーンBB*のシナリオ(**主にデータ**、少量の手続き:マップ、クリーチャー、NPC、特別なアイテム設定、トリガ、秘密の情報。必要に応じてローカル・ルール。)
ScenarioCC.lore	シナリオ・ライタ、GM、パワー・プレイヤー	*シナリオCC*(**殆どデータ**:マップ、配置済のクリーチャーやNPC、特別なアイテム設定、トリガ、秘密の情報etc.)

表 1に示すように全ての Lore ファイルで共通に利用できる標準的なデータを集めたのが”basic.lore”ファイルです。現在は MKS 単位系で長さ、重さ、時間、速度の単位のための単位型が定義されているだけですが将来的にはもっと役に立つ型や定数の定義が増える予定です。このファイルについては Lore 言語エンジン開発と並行して作成し、一緒に配布します。

ユーザ階層と役割分担の関係を模式的に示すため図 6には典型的な表 1に示した5種類のスタイルの Lore ファイルとそれに対応する作成者のタイプとの関連を示します。ゲーム・システムのルール定義から、世界設定、キャンペーン・シナリオ、通常のシナリオへと段階的にルールやフォームの定義といったプログラミング的な性質を持つ記述の量が減っていき、代わりにマップのデータやそこに仕掛けるトリガといったシンプルでオーサリング・ツールによるグラフィカルな表示・編集し易い記述の割合が増えていきます。これによって自然な形で図 6に示すようなユーザ階層に適合した形での導入が期待できます。

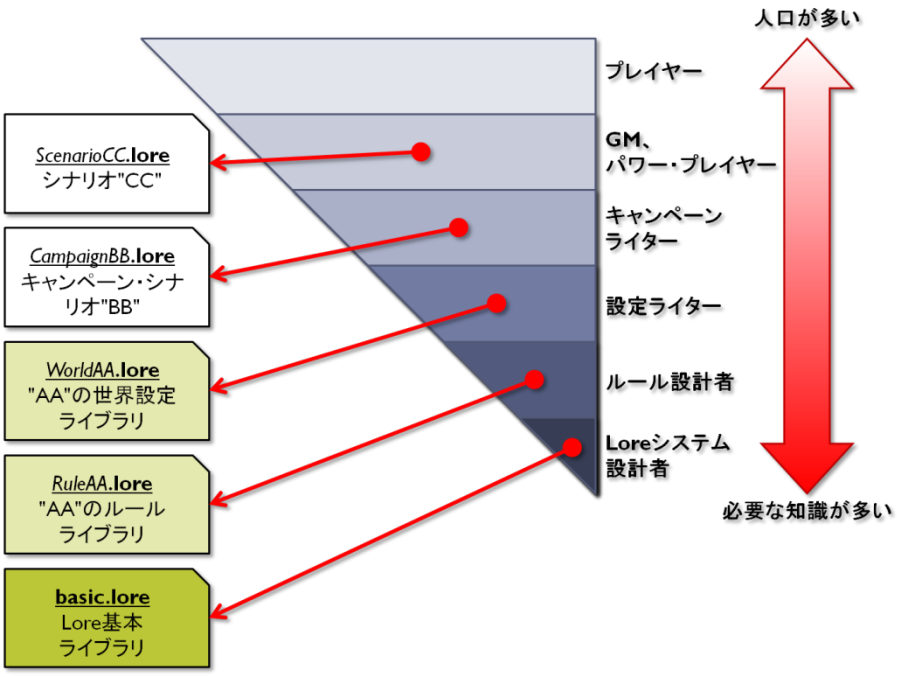

図　6：ユーザ階層

これらのファイル群は import 文を使って参照する関係にあります。そのような Lore ファイル間の参照関係と全ての Lore ファイルの形式を規定する Lore 言語エンジン・ライブラリ関係を模式的に示したものが次の図　7 となります。

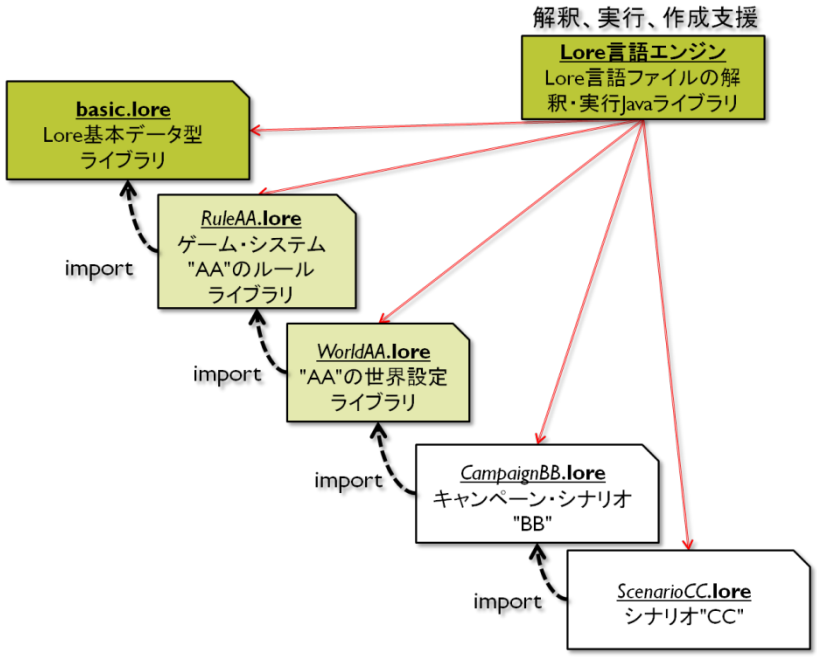

図　7：Lore 言語エンジン・ライブラリと各 Lore ファイルの関係

2 サンプルでみる Lore 言語

ここでは開発中の Lore 言語について、4 つの簡単なサンプルを元にシナリオやルールの記述の仕方を説明します。

2.1 Hello World

まずはプログラミング言語の伝統に従って”Hello World!”の例から。Hello World とは実行されると画面に Hello World!と表示して終了するだけの小さなサンプル・プログラムのことです。

```
docinfo{encoding="UTF-8", version="http://xgmtk.org/lore/1.0":url}
desc("Hello World"){"Hello Worldを表示するだけのサンプル"}
author("椎路 ちひろ"){"mailto:develop@xgmtk.org":url}

// 基本型のインポート
import "basic.lore":url;

section hello.world{
    /*
    開始直後（時刻0）にmainという名前のルールを適用するためのイベントです。
    実行可能なシナリオには最低一つのイベントが必要です。
    */
    first = {
        ticks = 0;
        priority = 0;
        handler=@{main;};
    }:Event;

    /*
    上記のイベントで適用されるルールです。
    GMのチャット・ウィンドウに"Hello world!"が表示されます。
    */
    rule main{
        // GMに向けてメッセージを送信します。
        GM.message("Hello world!");
```

```
    }
}
```

図 ８： HelloWorld.lore

この例は実行を開始すると GM の利用しているメッセンジャー・クライアントの画面に”Hello world!”という文字列を表示して終了します。

コメント ： 作成者、改訂者のための注釈

図 ８を見ると分かるように、”/*”から”*/”の間、そして”//”から改行までの間には自由にテキストを書くことができます。（このように記述される文字列を**コメント**と呼びます）。コメントはシナリオやルールを読む GM（ゲーム・マスター、レフェリーやキーパと呼ばれることもある）やルール作者・改訂者にルールやシナリオの各部について説明するためのものでオンライン・セッション・システムの動作には影響を及ぼしません。セッション中にプレイヤーに表示されることもありません。

メタ情報： 前準備（１）

まず図 ８最初の３行：

```
docinfo{encoding="UTF-8", version="http://xgmtk.org/lore/1.0":url}
desc("Hello World"){"Hello Worldを表示するだけのサンプル"}
author("椎路 ちひろ"){"mailto:develop@xgmtk.org":url}
```

これらはこのファイルそのものの情報（**メタ情報**といいます）を記述したものです。第１行目の docinfo 文はこの文書の文字コードが UTF-8 で Ver1.0 の Lore 言語仕様に従っていることを宣言しています。（現在の Lore 言語の実装では UTF-8 しかサポートしていませんので docinfo 文の内容はどのファイルでも同じくこのような記述となります。）２行目の desc 文（desc は「説明」を意味する description の略です）はこのファイルのタイトルと簡単な説明を記述してあります。３行目の author 文は著者情報で著者名と連絡先（URL で表記されるメールアドレス）が記入してあります。これらメタ情報は省略可能であり、必須ではありませんが、ルールやシナリオを流通させる場合や管理する場合に役立つので可能な限り記述するようにするのが良いでしょう。

インポート： 前準備（２）

メタ情報の行に続く部分がインポート文の並びです。

```
import "basic.lore":url;
```

import 文で他の Lore ファイルを URL により指定するとそのファイルに定義された様々なデータやルールを取り込み、自由に利用できるようになります。この例ではインポート文は 1 つだけで、Lore 言語の提供する基本的なデータやルールを利用できるよう basic.lore ファイルをインポートしています。

セクション ： 前準備（３）

インポート文に続く部分がルール記述あるいはシナリオ記述の本文となります。

この例では section 文が一つです。section 文は以下のように、section というキーワードに続けて名前（ここでは hello.world）、名前の後の”{”から最終行の”}”までがこの**セクション**の本体となります。

```
section hello.world{
＜本体の内容省略＞
}
```

セクションに指定する名前は自由につけて構いませんが、識別子（例えば”hello”）あるいは識別子を”.”で連結したもの（例えば”hello.world”）でなければなりません。**識別子**とは文字、数字とアンダースコア”_”の並びから成る文字列で先頭が数字で始まらないものです（識別子には日本語も利用できますが、個人的にあまりお勧めはしません。）。英字の大文字・小文字は区別されます。空白文字（空白や改行タブ）が紛れ込まないよう注意してください。

section名はsectionに含まれる様々な要素へのアクセスに利用することになります。別の言い方で言うと複数のファイル間で名前の重複を避けるための仕組み [2]です。このため後述するimport文により取り込んで利用するルール/シナリオ全体で重複しないように選ぶ必要があります。

重複しないようにするためにはルール・シナリオの作者が管理するインターネットドメイン名の情報を組み込んだ section 名にすることが考えられます。例えば筆者の場合であれば上述のような”hello.world”とする代わりに xgmtk.org を逆順にして組み込んで”org.xgmtk.hello.world”とするといったことです。これは他の、例えばプログラミング言語でもしばしば利用される命名方法ですが、インターネットドメイン名には識別子には使

[2] 他のプログラミング言語では名前空間、パッケージなどとも呼ばれます。

えない”-“（マイナス、ハイフン）が許される一方で識別子では許されている”_”（アンダースコア）が許されないなど識別子とインターネットドメイン名の規則は異なる部分があるので注意してください。

今回の例にはありませんがセクションは入れ子（セクション内でさらにセクションを定義する）が可能です。例えば外側のセクションの名前が outer で内側のセクションの名前が inner で inner セクション内の target という名前の対象にアクセスするとした場合、outer.inner.target という修飾された名前を利用することができます。また outer.inner というセクションを定義することとouterセクションの中にinnerセクションを記述することは同じ効果です。

またセクション内の各要素の順序は自由であり、どのような順序で並べても動作が変わることはありません。またセクションの順序自身も自由です。ファイル中で後ろにある要素の名前を前の方の要素で参照することには何の問題もありません[3]。文書としての作成しやすさ、読みやすさに従って自由に配置してください。

次は section の中を見てみます。この section は 1 つのイベントと 1 つのルールというたった 2 つの要素から成ります。

最初のイベント、そしてフォーム

1 つめの要素は**イベント**です。Lore 言語ではイベントによって様々なことが起こり、それにつれてゲーム内の時間が経過していく仕組みになっています。その辺りの仕組みについては後述のミニルール＆ミニシナリオの例で詳しく述べます。実行可能なシナリオには全ての基点として最低限一つのイベントが必要です。HelloWorld.lore では hello.world セクション最初の変数宣言文がそれです：

```
first = {
    ticks = 0;
    priority = 0;
    handler=@{main;};
}:Event;
```

この文はfirstという名前のイベントを一つ定義しています。イベントの名前もセクション

[3] 厳密には値の定義をする際に循環することは許されていません。つまり変数 A の値を計算するのに変数 B の値が必要で、B の値を計算するのに A の値が必要…といったことはできません。ルールの適用、form の定義の循環は可能です。

の名前と同じく識別子でなければなりません。イベントはフォーム（form）という仕組み[4]を使って定義されています。フォームは名前のついた 1 つ以上の値を組にしたものです。この組になっている値のそれぞれはフォームの**メンバ**と呼びます。またformの仕組みを使って定義された値の組全体からなる値は**オブジェクト**と呼びます。イベントを定義するにはEventという名前で定義されているformを利用します[5]。Eventにはticks、priority、handlerという三つのメンバがあるので上記のように３つの値をそれぞれ指定することで値（オブジェクト）を定義できます。ここで示したfirstのように値に割り当てた名前は**変数**と呼ばれます。

この first イベントはゲーム内でそのイベントが発生する時刻(時刻を表す ticks が 0 で、優先順位を示す priority も 0）に、後述する main という名前のルールを適用するためのイベントです。

ticksとpriorityは実行される時刻と優先順を示す整数値、handlerにはイベント発生時に実行する処理を記述します。ここではmainルールを適用するように記述しています[6]。ルールについては後で詳しく述べます。ticks属性に指定されている整数はシナリオ開始時からの経過時刻によってゲーム内での時刻を表し、特に 0 は実行開始時を表します。

たった１つのルール

その次、2 つめの要素は rule（ルール）です。:

```
rule main{
    // GMに向けてメッセージを送信します。
    GM.message("Hello world!");
}
```

rule 文は rule というキーワードに続けてルールの名前（識別子でなければなりません)、次いで”{“と”}”に挟まれた本体で構成されます。このルールの本体は//で始まる１行コメントと” `GM.message("Hello world!");`”がただ１行だけ書かれています。GM は Lore 言語があらかじめ用意する変数です。このルールが適用されるとオンライン・セッション・システムはセッションの開設者＝GM を知っているので、GM の PC 等で動いているクライアント・ソフトウェア（以下、クライアントと呼ぶ）へ向けて指定されたテキスト（ここで

[4] お気づきの方もおられるかと思いますが、form は多くのプログラミング言語で言われるところの class です。
[5] Event フォームは Lore 言語エンジンにあらかじめ組み込まれているフォームです。
[6] ルールとは多くの言語で言うところの値を返さない関数（「手続き」とも呼ばれる）で、@{…}は値としての関数を表す、関数型言語で言うところのラムダ記法です。

は”Hello world!”）を送信し、通信を受け取ったクライアントはチャット画面に”Hello world!”というメッセージを表示します。

結果として誰かがこの Lore ファイルをオンライン・セッション・システムにアップロードしてオンライン・セッションを開始すると、開始直後の時刻 0（優先順位も 0）に、登録されている first イベントが実行され、handler に指定されたアクションである main ルールが適用されます。main ルールは GM（セッションを開始した人）端末で動いているクライアントのチャット画面に”Hello World!”というメッセージを残して終了します。そして他にイベントがないのでセッションは即座に終了します。

2.2 ミニマムなシナリオ

ではシナリオの記述例に進みたいと思います。scenario.lore について説明します。非常に長い（特にマップ・データの定義を行うジオメトリについての部分が）なのでまずは概要を述べます。

このシナリオ・セクションは 8 個のオブジェクトから成っています。その内訳は以下の通りです：

- イベント（Event 型フォーム）×1 個…シナリオの開始時点を表現
- エリア（Area 型フォーム）×2 個…マップに相当
- アクター（Actor 型フォーム）×1 個…キャラクタのデータ
- アイテム（Item 型フォーム）×1 個…アイテムのデータ
- トリガ（Trigger 型フォーム）×3 個…マップ上などに仕掛ける仕掛け

アクターは TRPG で言うところのキャラクタです。コンピュータ関連では文字型をキャラクタ型と呼ぶことが多く紛らわしいので Lore 言語ではアクターとしました。

このようにシナリオは多くの場合こういったフォーム型の値を適切に配置することで記述できます。シナリオ固有のローカル・ルールを制定するのでない限り、オブジェクトを適切な位置に配置し、開始時点を表現するイベントを 1 つ置けばシナリオを記述することができます。

メタ情報、インポート、セクション

冒頭部分を図 １０に示します。

```
docinfo{encoding="UTF-8",version="http://xgmtk.org/lore/1.0":url}
desc("シナリオ・サンプル"){"Lore 言語のサンプルコード、シナリオの記述例"}
author("椎路 ちひろ"){"Chihiro_Shiiji@xmpp.xgmtk.org":jid,
    "mail:develop@xgmtk.org":url}
history("2013-07-15T00:00:00+09:00":date){reviser="椎路 ちひろ",desc="作成"}
history("2013-11-05T00:00:00+09:00":date){reviser="椎路 ちひろ",desc="改訂"}
history("2013-12-22T00:00:00+09:00":date){reviser="椎路 ちひろ",desc="改訂"}
```

図 ９：scenario.lore ファイル冒頭

docinfo文、desc文、author文は前節のHelloWorld.loreの例で説明したとおりです。ただauthor文の連絡先が一つ増えています。連絡先はJIDかURLであればいくつでも追加することができます。URLの場合は””で囲ったURLの後ろに”:url”を追加し、JIDの場合は””で

囲ったJabbar ID(XMPPのユーザアドレス)[7]の後ろに":jid"を追加して表します。複数の連絡先はカンマ","で区切ります。

その後に 3 行あるのは history 文です。作成、改訂などこの Lore ファイルの更新履歴を記録する文です。"history"キーワードに続けて"("")"内に日付け、"{""}"内に改訂者名を表す文字列（reviser）とその説明の文字列（desc）を記述します。日付は""で囲った日付を表す文字列（ISO 8601 形式）の後ろに":date"を追加して表します。

```
// 基本型のインポート
import "basic.lore":url;
// ルール定義のインポート
import "rules.lore":url;
```

図 １０：import 文

import 文で他の Lore ファイルを URL により指定するとそのファイルに定義された様々なデータやルールを取り込み、自由に利用できるようになります。この例ではインポート文は２つで、Lore 言語の提供する基本的なデータやルールを利用するための basic.lore ファイルと、シナリオの記述にゲーム・システムのルールを利用できるよう、後述する rules.lore ファイルをインポートしています。

このシナリオで定義される全ての要素は scenario セクションの中にあります。以下で順に見ていきましょう。

最初のイベントとイベント駆動による反復

scenario セクションの冒頭ではシナリオの開始のためのイベントが 1 つ宣言してあります。

```
initial = {
    ticks=0;
    priority=0;

    /* 初期化処理。 */
    handler = @{
        rules.initial;
    };
```

[7] XMPP はメッセージング（Skype や LINE の仲間）のための標準プロトコルです。Jabber ID（JID）はそのユーザを識別できるアドレスです。ユーザを識別する JID は「ユーザ名@XMPP ホスト名」の形式になるので見掛け上メールアドレスに似ています。

```
    }:Event;
```

図 １１：最初のイベント

イベントそのものは HelloWorld の例と殆ど同じです。違いはこのイベントの名前が initial であること、そしてこの例においては呼んでいるルール initial が同じセクション内でなく後述 rules.lore ファイルで定義されている rules セクション内にあるので rule.init のようにセクション名で修飾された名前で指定してあることだけです。この rules.init ように他セクションにある変数やルール等は”.”で区切ってセクション名と利用したいものの名前を区切って指定します。rules.init ルールは適用されると、プレイヤーに行動予定を尋ねそれに応じて行動に必要な手順や行動の結果を適用するために複数のイベントを作成します。

後の 2.3 節「ミニマムなルール」(p.36) でも述べるように一連のイベントでプレイヤーの行動が一段落すると、その最後のイベントで次の行動予定を尋ねて新たなイベントを登録するようにルールが書かれています。こうして「行動予定を尋ねる→（イベントが登録される）→（イベントが実行される）→行動の結果が確定する→（最初に戻る)」が延々と繰り返されることになります。このルールとシナリオは特に終わりがないので強制的に中断するまでそのサイクルが繰り返されます。強制的に中断するとセッションの情報が保存されたのちセッションは中断されます。セッションの情報の保存については 2.4 節「状態の保存」(p.50)で述べます。

「とある地方」 ： マップ（１）…ヒアドキュメント形式と XML 文字列

次の要素（図 １２）はこのシナリオに 2 つある内の１つめのマップ情報です。このエリア「とある地方」には実はたった２つしか地点がありません。「スタート地点」と「ミニマムダンジョン所在地」です。

```
    region = {
        location = "/[0]":loc;
        description = "とある地域";
        environment = Environment.inherited;
        geometry = [=[
<?xml version="1.0" encoding="UTF-8"?>
<geometry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.xgmtk.org/Lore/Linked Linked.xsd"
    xmlns="http://www.xgmtk.org/Linked"
    scale="1[km]">
    <node label="start_point">
```

```
        <description>
            <plain>スタート地点</plain>
        </description>
        <link to="dungeon" distance="4"></link>
    </node>
    <node label="dungeon">
        <description>
            <plain>ミニマムダンジョン所在地</plain>
        </description>
        <link to="start_point" distance="4"></link>
    </node>
</geometry>
]=]:xml;
    }:Area;
```

図 １２： マップ情報「とある地方」

この図 １２の region と名付けられたエリアの最初の３つのメンバはこのエリア全体の情報を指定します：

```
location = "/[0]":loc;
description = "とある地域";
environment = Environment.inherited;
```

location メンバはこのマップが上位のエリアのマップのどこに位置するかを示します。「上位のエリア」の意味するところは後述します。description メンバはこの地域についての簡単な説明をテキストで記述します。environment メンバは高度や気候、方位といったこのエリア全体の環境に関する情報を定義します。Environment.inherited は特別な値で、このエリアの上位のエリアの環境を引き継いでいることを示します。

Lore言語のマップ・データは上記のようなAreaと名付けられたform（以降エリア [8]と呼ぶ）で表現されます。各エリアの表現はGUIのリッチさとオーサリングの手間とのトレードオフや情報の粗密のメリハリに配慮して複数のスタイルがあり必要に応じて選択することができます。

エリアはシナリオとルール全体では複数のスタイルが混合して利用されることを想定しています。複数のタイプのエリアは様々なスケールを設定できます。図 １３に示すように

[8] Area フォームは Lore 言語エンジンにあらかじめ組み込まれているフォームです。

エリアは木構造により全体として階層化され、木構造のノード毎に異なるタイプ、スケールのエリアが混在することになります。

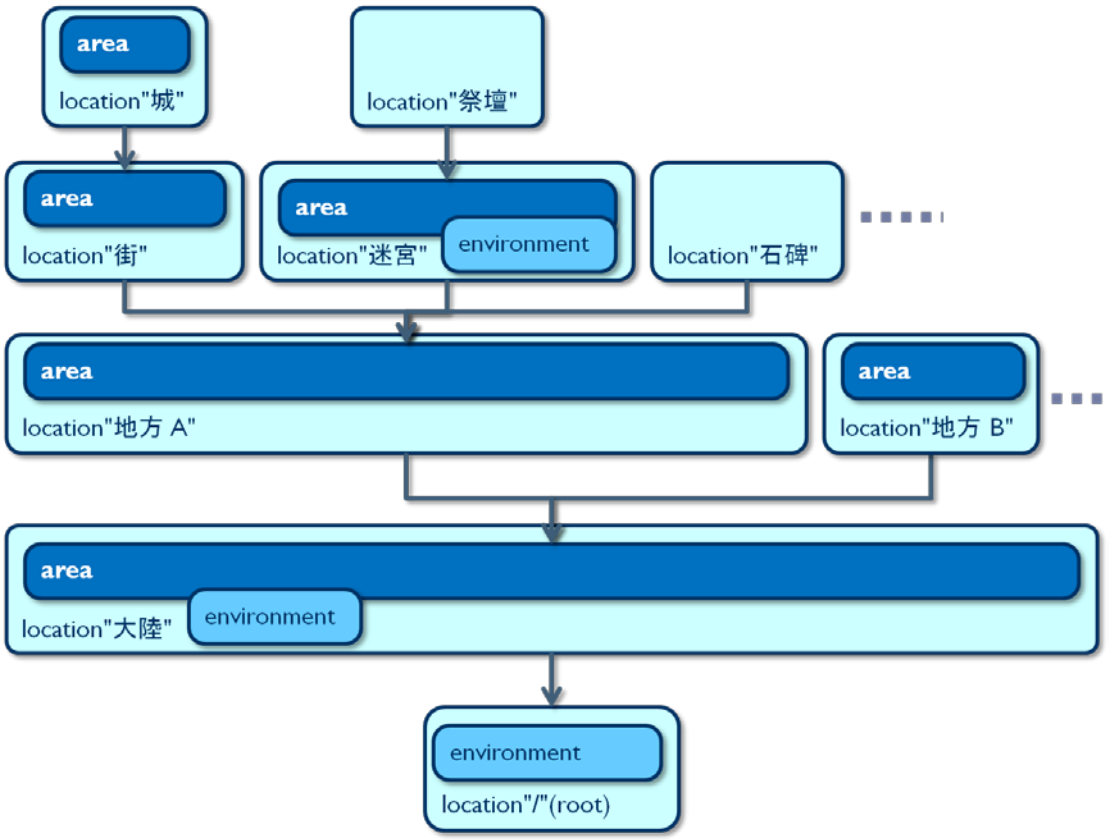

図 １３:ロケーション型とロケーションパス

このような階層的なエリアの構造内でロケーションを一意に指すのが**ロケーションパス**です。ロケーションパスは”/”を木構造の根（ルート）としその後ろに根に近い方から順に木構造の特定のノードに向かって各エリアの**ラベル**（label 要素に指定される）を並べ、間を”/”で区切ったもので特定のエリアを指定し、その後ろに省略可能なオプションとして”[“と”]”で囲ったそのエリアのジオメトリ内の特定の位置を示す「**座標**」付加したものです。座標がないロケーションパスはエリア全体を指します。ロケーションパスの値はそれを表す文字列を二重引用符で括って後ろに”:loc”を付加して表します。図 １２の region エリアの例では location メンバの値に指定されている"/[0]":loc がそれです。この時木の根に当たるノードである”/”に近い方が「上位のエリア」となります。(「上位」といいつつ図 １３では下の方が「上位」になっています。)

各エリアの location メンバは自身の位置が上の階層のマップのどの位置にあるかを示します。location メンバはエリア以外にアクター（actor 要素）やアイテム（item 要素）にもあり、ロケーションパスによってそれらの位置を指定するのに使われます。

エリアの第 4 のメンバ、geometryメンバはこのエリアの地形情報を記述する**XML文字列**です。XML文字列はXML文書を構成する文字列の後ろに":xml"を付加したものです。ここでは改行や二重引用符を記述したいのでここまで出てきた二重引用符による文字列ではなく**ヒアドキュメント**表記の文字列を使っています。ヒアドキュメント表記では"[=["から"]=]"までが文字列となり、その間二重引用符も改行も自由に使うことができます。もしヒアドキュメント文字列中に"]=]"を書きたいときは開始記号を"[==["終了記号を"]==]"とすることができます[9]。XML文書を直接XML文字列として記述してあります。

XML 文書は XML 形式で記された構造を持った文字列、あるいはその文字列を内容とするテキスト・ファイルを指します。XML 文書の規則は簡単で、XML 文書はその文字列が XML 文書であることを宣言する最初の 1 行（**ドキュメント宣言**）を除いて**要素**（element）が入れ子になった階層構造しています。各要素は名前を持っており、その名前を XYZ とすると**開始タグ**は<XYZ>であり**終了タグ**は</XYZ>となります。各要素はその開始タグから終了タグの間に別の要素（**子要素**、child element）や任意のテキストといった**内容**（contents）を持つことができます。そして一番「外側」即ち最初に開始タグが来て、最後に終了タグが来る要素は特にルート要素（root element）と呼ばれます。その要素が子要素やテキストといった内容を持たない、つまり開始タグの直後に終了タグが来て<XYZ></XYZ>のようになっているときは<XYZ/>と略記することができます。また要素は名前付きの値である**属性**（attribute）を 0 個以上持つことができ、例えば abc という名で値が"def"である属性を XYZ が持っていれば<XYZ abc="def"></XYZ>や<XYZ abc="def"/>と書くことができます。

geometry メンバに指定された文字列は XML 文書なので最初の行は以下のようなドキュメント宣言となります：

```
<?xml version="1.0" encoding="UTF-8"?>
```

このドキュメント宣言は大体決まり文句と思って頂いて構いませんが、一応 XML 文書の規格のバージョンが 1.0 であることと、文字コードが UTF-8 であることを宣言する意味があります。

geometry メンバに指定できる XML 文書のルート要素は geometry 要素となります。geometry 要素の開始タグを示します：

```
<geometry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

[9] 終了記号の"]"と"]"に挟まれる"="の数は開始記号の"["と"["に挟まれている"="の数と等しくありさえすれば自由に決めることができます。このアイディアは Lua 言語のヒアドキュメントの記法に倣ったものです。

```
xsi:schemaLocation="http://www.xgmtk.org/Lore/Linked Linked.xsd"
xmlns="http://www.xgmtk.org/Linked"
scale="1[km]">
```

geometry要素のxmlns属性 10にはこのジオメトリのスタイルをURIで指定します。ここでは”http://www.xgmtk.org/Lore/Linked”が指定されているのでグラフ構造で表現されるgeometryを持つことが指定されています。xmlns:xsi属性とxsi:schemaLocation属性はこのgeometry要素の構造がXML Schemaで定義されていてその定義が” " Linked.xsd"であることを示しています。xmlns:xsi属性はいつもこの値、xsi:schemaLocationはジオメトリのスタイルに対応するXML SchemaファイルがLinked.xsdであることを示しています。一般にLore言語ではジオメトリのスタイルの名が*X*であるとするとスタイルのURIは ”http://www.xgmtk.org/Lore/*X*” という形になりそのXML Schemaファイルの名は*X*.xsdとなります。

scale 属性はこのマップ・データの縮尺情報として基本となる長さを指定します。ここでは単位付きの数値型により 1km が基本となる長さとして指定されています。

さて、この geometry 要素の中身が実際の地形データとなります。Linked スタイルのジオメトリはリンクされた各点によるグラフで表現されるジオメトリです。XML 的にはジオメトリは node 要素（ノード）の集まりです。この図 １２の例の場合ノードは 2 つだけです：

```
<node label="start_point">
    <description>
        <plain>スタート地点</plain>
    </description>
    <link to="dungeon" distance="4"></link>
</node>
<node label="dungeon">
    <description>
        <plain>ミニマムダンジョン所在地</plain>
    </description>
    <link to="start_point" distance="4"></link>
</node>
```

10 正確に言えばこれはXMLの名前空間宣言なのですがXMLの名前空間の説明は長くなるのでここは単に xmlns 属性にはジオメトリのスタイルを表す URI（ここでは URL）を指定するとしています。

ノードの「座標」は label 属性に指定される識別子で表され、この例ではそれぞれ”start_point”と”dungeon”となります。各ノードにはユーザに表示される情報としてデスクリプションを付加することができます。

各ノードは linke 要素（リンク）を持ち、これでノード間の結びつきを表現します。リンクはその link 要素を含むノードから to 属性で指定されたノードへの一方通行の繋がりであり、2 点間で双方向にするには互いに相手へのリンクを持つようにします。各リンクには distance 属性に数値を指定することで距離を指定できます。distance 属性に指定された数に geometry 要素の scale 属性で指定された距離を掛けたものが実際の距離になります。

上記の例では start_point から dungeon へ、また dungeon から start_point へ相互にリンクが張られていて distance は 4、geometry 要素の scale 属性の値は 1[km]なので開始地点とダンジョン所在地はどちらからどちらへの道のりも 4km の距離ということになります。

ちなみに distance 属性が省略された場合の値は 1、即ち geometry 要素の scale 属性で設定された値と等しい距離となります。distance 属性に指定された値について、例えば二点間の相互リンクで行きと帰りの数字が等しいかどうかといったような幾何学的なチェックはされません。直接リンクされていない各点間の距離はグラフの最短経路から計算されます。正確にいえば Linkedna ジオメトリで計算される距離は移動経路距離であって幾何学的な最短距離ではありません。しかしこのジオメトリは元々情報の密度が極めて粗いマップに用いられる大変大雑把なものであり、あまり精密な距離の表現を期待するものではありません。

Linked スタイルのジオメトリは往年のテキスト・アドベンチャーで使われていたような離散的なマップを表現することが主な目的となります。想定されるグラフィカルな表示法はありませんが、デスクリプションに HTML テキストを指定することができ、それによって装飾されたテキストや画像を表示させるといったようなことができます。このジオメトリのビジュアルは極めて貧弱ですが、それ故にこのジオメトリを持つエリアの作成は非常に簡単ですので、シナリオの試作に利用したり、シナリオ上重要なエリア間の「糊」（シナリオ上の重要な地点間を結ぶ情報量の少ないエリアとして用いる）のように利用したりできるでしょう。このシナリオの例では開始地点とダンジョンの間を結ぶ役割を果たしています。

「とあるダンジョン」 ： マップ（２）…外部 XML 文書

次の要素（図 １２）はこのシナリオに 2 つある内の 2 つめのマップ情報となる dungeon と名付けられたエリアです。このマップは
マス目に区切られた形式で表現されるマップです。この例では５×５のマス目に区切られたダンジョンを表しています。

```
    dungeon = {
        location = "/region[dungeon]":loc;
        description = [=[Sectioned2D型ジオメトリ(スクエア、5×5)のエリア。
ダンジョンには入口（座標[4,2]）と祭壇（座標[1,2]）がある。
祭壇の升目だけは30cmほど周囲より高い。
スケールは5mとするので升目の1辺が5mとなります。
]=];
        geometry = ("dungeon.xml":url).loadXML;
        environment = {
            height = -5[m];
            rotation = 30[degree];
        }:Environment;
    }:Area;
```

図 １４： とあるミニマムなダンジョン

Area 型の値の記述方法は先ほどの図 １２の例と同じです。dungeon エリアは region の下位エリアで region 内の”dungeon”というノードに位置するため location メンバには”/region[dungeon]”:loc という値が指定されています。説明文が長いため description メンバには通常の文字列ではなくヒアドキュメント形式で説明文が書いてあります。

先ほどの「とある地方」の図 １２の例との大きな違いとしては、ジオメトリは別ファイル（**外部 XML 文書**）になっていて、セッション開始時に指定された URL から読み込むようになっています。URL 文字列の loadXML というアクセサがそれを実現します。アクセサとはアクセサ名（ここでは loadXML）の左にある”.”の左に指定された値（ここでは "dungeon.xml":url）の属性にアクセスし、その値を返すことができる特別な種類のルールです。URL の loadXML 属性の値は URL で指定された XML ファイルの内容である XML 文書そのものです。

また environment メンバが付加され、そこに height が指定されてその要素の値に-5m が指定されているので、このエリアは親のエリアから地下へ 5 メートルの深さにあることになります。通常はジオメトリで記述されるマップは上が北ですが、この例では environment メンバの rotation メンバ（向き）に 30 度を指定したので上空から見て時計回りに 30 度回転されています。

別ファイルの中身は図 １５のようになります：

```
<?xml version="1.0" encoding="UTF-8"?>
<geometry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.xgmtk.org/Lore/Sectioned Sectioned.xsd"
```

```
xmlns="http://www.xgmtk.org/Lore/Sectioned2D"
scale="5[m]" type="4">
<terrains>
    <element name="floor" />
    <element name="wall" />
</terrains>
<tile terrain="floor" src="floor.jpg" />
<tile terrain="wall" src="wall.jpg" />
<line>
    <section terrain="wall" prohibited="true" />
    <section terrain="wall" prohibited="true" />
    <section terrain="wall" prohibited="true" />
    <section terrain="wall" prohibited="true" />
    <section terrain="wall" prohibited="true" />
</line>
<line>
    <section terrain="wall" prohibited="true" />
    <section terrain="floor" />
    <section terrain="floor" />
    <section terrain="floor" />
    <section terrain="wall" prohibited="true" />
</line>
<line>
    <section terrain="wall" prohibited="true" />
    <section terrain="floor" height="0.3(m)">
        <description>
            <plain>祭壇</plain>
        </description>
    </section>
    <section terrain="floor" />
    <section terrain="floor" />
    <section terrain="floor">
        <description>
            <plain>入口</plain>
        </description>
    </section>
```

```
    </line>
    <line>
        <section terrain="wall" prohibited="true" />
        <section terrain="floor" />
        <section terrain="floor" />
        <section terrain="floor" />
        <section terrain="wall" prohibited="true" />
    </line>
    <line>
        <section terrain="wall" prohibited="true" />
        <section terrain="wall" prohibited="true" />
        <section terrain="wall" prohibited="true" />
        <section terrain="wall" prohibited="true" />
        <section terrain="wall" prohibited="true" />
    </line>
</geometry>
```

図 １５： とあるミニマムなダンジョンのジオメトリ

このシナリオではこの図 １４のエリアは geometry 要素の xmlns 属性に http://www.xgmtk.org/Lore/Sectioned2D が指定されているので 2 次元マス目（ヘックスあるいはスクェア）スタイルのマップとなります。

```
<?xml version="1.0" encoding="UTF-8"?>
<geometry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.xgmtk.org/Lore/Sectioned Sectioned.xsd"
    xmlns="http://www.xgmtk.org/Lore/Sectioned2D"
    scale="5[m]" type="4">
```

このエリアのジオメトリはエリアの geometry 属性とジオメトリ要素の xmlns 属性で示されているように SectionedD2 スタイルです。SectionedD2 スタイルはマス目（スクエアやヘックス）が規則正しく並んだものとして表現されるエリアのためのジオメトリです。マス目の性質を表すため Sectioned2D スタイルの geometry 要素には固有の属性として type があります。type 属性に指定された値が 4 なら縦横 4 方向へ移動が可能なスクエア、6 ならヘックス、8 なら縦横斜めの 8 方向へ移動が可能なスクエアとなりますが、この例では 4 となっています。scale 属性は 5[m]なのでマス目の中心と隣のマス目の中心との距離は 5[m]となります。

geometry 要素内の先頭の子要素は terrains 要素です：

```
<terrains>
    <element name="floor" />
    <element name="wall" />
</terrains>
```

terrains 要素はマス目毎にあり得る地形の種類を示すを持ちます。terrians 要素は name 属性のみを持つ element 要素の並びによりマス目毎の「**地形**」の種類を示す識別子を列挙します。この例では地形は wall と floor の二種類です。このシナリオでは wall は壁、floor は普通の床を想定しています。

続いてこの例では tile 要素が並びます：

```
<tile terrain="floor" src="floor.jpg" />
<tile terrain="wall" src="wall.jpg" />
```

Sectioned2D ジオメトリは terrians 子要素の次の子要素として一つの visual 要素あるいは複数の tile 要素のどちらかを持ちます。visual 要素と tile 要素はマップの見えかたを定義します。tile 要素があるとき、表示はマス目に設定された地形の値に応じて選ばれるアイコンを各マス目に表示してマップ画像を作成します。tile 要素は対応する地形を指定する terrain 属性とアイコンに利用する画像の URL を指定する src 属性を持ちます。visual 要素あるときは一枚の絵でマップ全域を表示し移動等の時だけマス目を意識します。visual 要素の持つ src 属性はこの 1 枚絵の画像ファイルの URL です。

tile 要素、あるいは visual 要素に引き続く line 要素（ライン）の並びがこのジオメトリの本体です。ラインには各マスを表す section 要素が並びます。全てのマスには 2 つの整数の組[x,y]で表現される座標が対応付けられ、特に指定がない場合左上が原点[0,0]で右に向かって x 座標の値が増え、下に向かって y 座標が増えます。この例にはラインが 5 つもあるので代表的な例として 3 つめのラインを示します：

```
<line>
    <section terrain="wall" prohibited="true" />
    <section terrain="floor" height="0.3(m)">
        <description>
            <plain>祭壇</plain>
```

```
            </description>
        </section>
        <section terrain="floor" />
        <section terrain="floor" />
        <section terrain="floor">
            <description>
                <plain>入口</plain>
            </description>
        </section>
    </line>
```

このラインは y 座標が 2 であるマス目の集まりです。ライン内には 5 つの section 要素（マス）が並んでおり座標[0,2]から[4,2]のマスに対応しています。[0,2]のマスの terrain 属性は wall ですので壁を表現します。また prohibited 属性に true が指定されているので立ち入り禁止です。[1,2]～[4,2]のマスの床 terrain 属性は wall ですので床を表現します。その中で特に[1,2]に当たるマスには height 属性に 0.3m が指定されているため他のマス目より 30cm 高いことになります。[1,2][4,2]のマス目にはそれぞれ「祭壇」「入口」というデスクリプションが指定されています。

この SectionedD2 スタイルのジオメトリは古典的に多くのストラテジー系のテーブル・ゲームで利用されてきていてユーザによく親しまれており、ユーザにとっては距離や方角が目測しやすいという特徴があります。ゲーム進行中にスクリプト言語により動的な生成をすることも簡単なため、戦略的性が重要な戦闘の場合に、一時的なエリアを生成して用いることもできるでしょう。ストラテジー系のゲームでおなじみの「地形効果」のようなものを表現することもできます。ただしエリア内のすべてのマス目に地形を設定していく作業があるので作成にはそれなりに手間がかかります。2 次元のマップとしては情報の密度が濃いのでシナリオ上の重要地域や重要な戦闘の舞台となりそうな場所で使用するとよいでしょう。またこのジオメトリはグラフィカルな UI を備えたマップエディタなどオーサリング・ツールがあれば作成がかなり容易になるでしょう。

キャラクタ（アクター）

このシナリオのセクションの次のオブジェクトは pc と名付けられた PC フォームのオブジェクトです。

```
pc = {
    name="Chihiro";
    player="Chihiro_Shiiji@xmpp.xgmtk.org":jid;
```

```
        location="/region[start_point]":loc;
        features=list<Featur>.empty;
        items=list<Item>.empty;
        knownSecrets=list<string>.empty;
    }: rules.PC;
```

図 １６： このシナリオの唯一のキャラクタ（アクター）

rules.PC フォームは後述するように Lore 言語があらかじめ用意しているフォーム Actor を拡張したもので、この rules.PC フォームの場合、保持するメンバは Actor と同じです。Actor フォームは TRPG ではキャラクタとして親しまれているオブジェクトのためのフォームですが、”character”は”前述したように多くのプログラミング言語にある文字型と紛らわしいので Lore 言語では Character ではなく Actor という名前になっています。

ActorフォームにはGMやプレイヤーに表示する際の名前を指定する”name”メンバ、プレイヤーのチャット・アカウントのアドレスであるJID[11]を指定するplayerメンバ、アクターの現在位置を指定するlocationメンバ、各種能力値を格納するfeaturesメンバ、所有アイテムを格納するitemsメンバ、既知の秘密を表すknownSecretsがあります。

「とある地方」（/region）の”start_point”に配置するため、location メンバに指定されるロケーションパスは/region[start_point]となっています。

この例のルールは単純で能力値を一つも設定しないので能力値のリストは空のリストを表す list<Featur>.empty となっています。ここで Feature は Lore 言語があらかじめ用意しているフォームで能力値を表し、list<Feature>は Feature のリストを表します。

セッション開始時には所持品はないのでアイテムのリストは空のリストを表す list<Item>.empty となっています。ここで Item は Lore 言語があらかじめ用意しているフォームでアイテムを表し、list<Item>は Item のリストを表します。

セッション開始時には既知の秘密はないので既知の秘密のリストは空のリストを表す list<string>.empty となっています。ここで string は Lore 言語があらかじめ用意している文字列の型名で、list<string>は文字列のリストの型名です。knownSecrets はそのアクターが知ることになった数々の秘密を識別する文字列のリストとなっています。秘密については後述します。

アイテム

このシナリオのセクションの次のオブジェクトは chest と名付けられたアイテム・オブジェクトです。

---

[11] オンライン・セッション・システム XGMTK は XMPP を利用して作られているのでユーザは XMPP のユーザを識別する Jabber ID こと JID で識別できます。

```
chest = {
    location = "/region/dungeon[1,2]":loc;
    features=list<Featur>.empty;
    secrets = {
        {
            key = "secret0";
            mode = Secret.Mode.hide;
        }:Secret
    };
}:Item;
```

図 １７： このシナリオ唯一のアイテム

この例ではアイテムを「とあるミニマムなダンジョン」(/region/ dungeon）の祭壇（座標[1,2]）に配置するため、location メンバの値として指定されるロケーションパス/region/dungeon[1,2]が指定されています。

このシナリオが参照するルールではアイテムの属性が何も設定されていないのでアイテムの性質や能力を示すのに利用される先ほどのアクターの例と同じく features メンバは空です。

このアイテムは秘密のリストである secrets メンバに 1 つだけ Secret オブジェクトが入れられています。Secret オブジェクトは秘密の存在を示すオブジェクトです。ここでは mode メンバに Secret.Mode.hide という値が指定されています。このように Secret.Mode.hide が指定されている Secret オブジェクトが secrets メンバのリストに含まれる場合、そのアイテムはアクターから「見えない」ものとして扱われます。端的に言えば、プレイヤーが見ているそのアクターに対応するマップ画面上に表示されません。これがアクターから「見える」ようになるにはそのアクターが Secret の key メンバに指定された秘密識別文字列と同じ値を、knownSecrets のリストに持つ必要があります。言いかえると何らかのアクションの結果として knownSecrets に秘密識別文字列が追加される必要があります。

トリガ

このシナリオを締めくくる最後のオブジェクトはシナリオ上の仕掛けを表現する３つの trigger オブジェクト（**トリガ**）です。トリガとは条件を満たすある地点でアクターが特定のアクションを起こすと何かが起こる、という仕組みを実現するためのものです。条件が満たされてトリガが何かを引き起こすことをトリガが**発火**するといいます。３つのトリガの内、最初の２つはエリア間の移動のためのもので、最後の１つはダンジョン内に隠しておいた chest に関するものです。

```
dungeonEntrance = {
    location="/region[dungeon]":loc;
    effectiveRange = 0[km],...<1[km];
    actions={Action.search};

    // ダンジョンに入るためのイベントのハンドラ
    pcHandler = @(pc : rules.PC){
        // 指定された時間経過後に移動完了の処理をするための
        //イベントを登録します。
        pc.futureMove(1[R], "/region/dungeon[1,4]":loc);
        pc.message("ダンジョンに入ります。");
    };
}:rules.PCTrigger;
```

図 １８： ダンジョンへの入り口

dungeonEntrance 変数に割り当てられているのは「とある地方」から「とあるミニマムなダンジョン」へエリア間移動のためのトリガ・オブジェクトです。ダンジョンの地点で「探す」をすると次のラウンドにダンジョンに入るという効果を実現しています。

rules.PCTrigger フォームは Lore 言語が提供する Trigger を拡張して作られたフォームです。その定義は rules.lore にあります。rules.PCTrigger フォームはトリガの置かれた位置を示す location、トリガが発動する有効距離の範囲を示す effectiveRange、アクターがどのアクションを起こした際にトリガを発火させるべきかを示す actions, トリガが発火した際の処理を記述する pcHandler という 3 つのメンバがあります。

この例の場合、location メンバには「とある地方」にある 2 つの地点のうち、「ダンジョンのある地点」を指定するロケーションパス"/region[dungeon]":loc が指定されています。effectiveRange には範囲型の値で 0km 以上かつ 1km より短い範囲を示す 0[km],...<1[km] という値が指定されています。これにより「とある地方」に存在する互いに 4km 離れた 2 点の内、アクターがダンジョンのある地点にいる間に限りこのトリガがチェックされます。

actionsが空のリストである場合、トリガは範囲内に立ち入ったアクターがいると自動発火します[12]。この事例ではaction要素が指定されているので、立ち入ってかつ指定された値で識別されるアクションを実行したアクターについてだけトリガが発火します。actionsメンバのリストにはActions.searchが格納されているので、上記の位置にいる間にアクターが「探す」というアクションを実行するとトリガが発火します。

[12] 正確に言うと actions が空である場合、アクターの位置が更新される毎にトリガの範囲内にいるかどうかがチェックされ、範囲内ならば発火します。

pcHandler 属性はトリガが発火した際に呼び出される処理を記述します。イベントの handler の例と似ていますが、違いとしてはトリガの pcHandler ではハンドラの処理を記述する”{“から”}”の間に書かれる処理において PC フォームの pc という変数にアクセスできます。そのことは”@”と”{“の間にある”(pc：PC)”で指定されています。この変数 pc はトリガを発火させたアクターを表しています。

この handler の処理ではまず後述するように PC オブジェクトである pc の後ろの”.”に続けて PC フォームで定義されているルールの名を記す futureMove ことで、PC オブジェクトに対して futureMove ルールが適応されます。futureMove ルールは追加の引数として所要時間と移動後の位置を示すロケーションパスを必要とするの”(“”)”で括って二つの値を指定します。これによって 1[R]経過後にアクターの位置がダンジョン内入ったところ（"/region/dungeon[1,4]":loc）になるように書きかえるイベントを登録されます。次いで、pc.message ルールにより"ダンジョンに入ります。"というメッセージがアクターに送られ、プレイヤーのチャット画面に表示されます。

```
dungeonExit = {
    location="/region/dungeon[1,4]":loc;
    effectiveRange=0[m],...<5[m];
    actions={Action.search};

    // ダンジョンを出るためのイベントのハンドラ
    pcHandler = @(pc：rules.PC){
        // 指定された時間経過後に移動完了の処理をするための
        // イベントを登録します。
        pc.futureMove(1[R], "/region[dungeon]":loc);
        pc.message("ダンジョンから出ます。");
    };
} : rules.PCTrigger;
```

図 １９： ダンジョンからの出口

dungeonExit 変数に割り当てられているのは「とあるミニマムなダンジョン」から「とある地方」へとエリア間移動のためのトリガ・オブジェクトです。ダンジョンの出口地点で「探す」をすると次のラウンドにダンジョンから出るという効果を実現しています。

この例の場合、location メンバには「とあるダンジョン」の出入り口のロケーションパス "/region/dungeon[1,4]":loc が指定されています。effectiveRange には範囲型の値で 0m 以上かつ 5m より短い範囲を示す 0[m],...<5[m]という値が指定されています。これにより「とあるダンジョン」内の[1,4]から 0m 以上 5m より近い位置、つまり[1,4]のマス目そのもの（このエリアの sacale が 5[m]に設定されているため隣のマスまでの距離は 5[m]です）に

アクターがいる間に限りこのトリガがチェックされます。

この事例では Actions.search が格納されているので、上記の位置にいる間にアクターが「探す」というアクションを実行するとトリガが発火します。

トリガが発火すると pcHandler に記述された処理が実行されます。この pcHandler の処理ではまず後述するルール futureMove が適応され 1[R]経過後にアクターの位置が「とある地方」にある 2 つの地点のうちダンジョンのある地点を指定するロケーションパス（"/region[dungeon]":loc）になるように書きかえるイベントを登録されます。次いで、pc.message ルールにより"ダンジョンから出ます。"というメッセージがアクターに送られ、プレイヤーのチャット画面に表示されます。

```
    chestFound = {
        location="/region/dungeon[1,2]":loc;
        effectiveRange=0[m],...<5[m];
        actions={Action.search};

        /*
        宝物発見のトリガのハンドラ。
        秘密情報をキャラへ渡すことで宝箱を見えるようにするとともに
        メッセージを表示します。
        */
        pcHandler = @(pc：rules.PC){
            // チェストに設定されている秘密情報をキャラに受け渡す
            pc.addKnownSecret(chest.secrets.get(0).key);
            /*
            ここでは html テキストをメッセージとして送ってみました。
            大きな文字と宝箱のイメージがチャット画面に表示されます。
            */
            pc.message([=[
<html>
    <body>
        <p>
            <big>宝物が見つかりました！</big>
            <img src="TresureChest.jpg" alt="（宝箱のイメージ）" />
        </p>
    </body>
```

```
</html>
]=]:html);
            };
    }:rules.PCTrigger;
```

図 ２０： 宝箱の発見

chestFound 変数に割り当てられているのは宝箱発見のためのトリガです。ダンジョンにある祭壇の位置で「探す」をすると発火します。

この例の場合、location メンバには「とあるダンジョン」の祭壇のロケーションパス "/region/dungeon[1,2]":loc が指定されています。effectiveRange には範囲型の値で 0m 以上かつ 5m より短い範囲を示す 0[m],...<5[m]という値が指定されています。これにより「とあるダンジョン」内の[1,2]から 0m 以上 5m より近い位置、つまり[1,2]のマス目そのものにアクターがいる間に限りこのトリガがチェックされます。

この事例では Actions.search が格納されているので、上記の位置にいる間にアクターが「探す」というアクションを実行するとトリガが発火します。

トリガが発火すると pcHandler に記述された処理が実行されます。この pcHandler の処理ではまず後述するルール addKnownSecret が適応されます。それにより宝箱 chest を「見えない」状態にしていた Secret の key メンバの値（chest.secrets.get(0).key で表される）がアクターの knownSecrets に追加され、宝箱がアクターから見える状態になります。次いで、pc.message ルールにより"宝物が見つかりました！"というメッセージと宝物の画像がアクターに送られ、プレイヤーのチャット画面に表示されます。この例ではメッセージを通常の文字列ではなくヒアドキュメントで HTML を記述した html 型文字列を使い、単なるテキストではなく装飾された文字と画像を送信しています。

以上でスタート地点から旅立って一つしか部屋のないダンジョンの祭壇で宝箱を見つけることができるシナリオが記述できます。このように典型的なシナリオは殆どオブジェクトの配置で記述できるためオーサリング・ツールで作成しやすい性質があることがわかります。詳細や手続き的な処理は全てルールの側に書かれています。

2.3 ミニマムなルール

前節のシナリオは実際にはそれだけではまだ動作しません幾つかの規則の記述が必要です。ルールは複数のシナリオで共有される共通の記述（各種のデータやルール）です。この部分を記述することで様々な TRPG のシステムを記述することになります。

ここでは「探す」、「移動する」と「寝る」の 3 つしか行動の選択肢がないミニマムなルールの記述である rules.lore の rule セクションについて説明します。HelloWorld.lore と

scenario.lore でメタ情報やインポート文、セクション文については述べたのでこの節では省略します。長いのでまずは概要を述べます。

このルール・セクションは以下の要素から成っています。その内訳は以下の通りです：

- 単位定義×１個
- 単位型の変数定義×１個
- 列挙定義×１個
- エリア（area オブジェクト）×1 個
- ルール定義×７個を含むフォームの拡張×１個
- ルール定義×１個
- ルール定義×１個、メンバ定義×1 個を含むフォームの拡張×1 個

これらについて以下順に見ていきます。

[R] ： 単位の定義

まずはゲーム内の基本的な時間単位であるラウンドの定義。ここでは 1[R]を 5 分とします。[min]は[sec]に基づいて basic.lore で定義されています。[sec]は時間の基本単位として basic.lore で定義されています。この定義により[R]をゲーム内の時間の単位として利用することができます。

```
[R]=5[min];
```

図 ２１： 単位の定義

このように特定の単位付き数値型から定義される一群の単位付き数値型は図 ２２のようにファミリを成します。同一のファミリに属する型は加算と減算の際に基底となった型に自動変換されて演算されます。

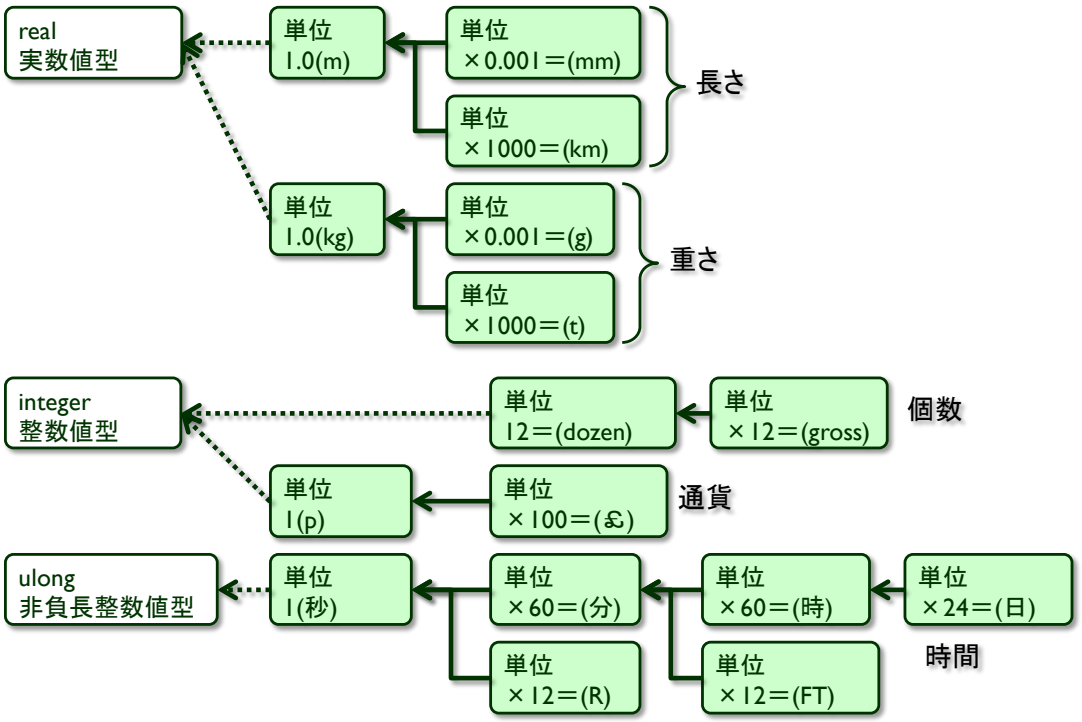

図 ２２：単位型ファミリの例

[m/sec]も basic.lore で定義されています。このように単位名には"/"や"^"を使うことができます。但し単なる識別用の文字列であり組立単位として解釈されることはありません。

standard_velocity：変数の定義

変数として歩く際の標準的な移動速度 `standard_velocity` を定義します。`[m/sec]`は速度の基本単位として `basic.lore` で定義されています。（秒速 `1.6m` は時速 `6km` 弱の早歩きくらいの速度なので長距離を移動するには早すぎるでしょうが…。）

```
standard_velocity = 1.6[m/sec];
```

図 ２３： 定数の定義

アクションの種類の定義

アクションの種類search / move / sleepを表す値を定義します[13]。それぞれの値は表示用の文字列captionを値として持っています。このルールは単純なのでこのようにたった 3 種類、探す、移動する、休むしかありませんが、実際のTRPGではもう少し沢山のアクションがあることでしょう。

```
enum Action{
    search{caption="探す";}
```

[13] プログラマの方はお察しの通りこれは列挙型です。Java の列挙型のように関連付けて複数の値を持たせることができます。

```
        move{caption="移動する";}
        sleep{caption="寝る";}
    }
```

図 ２４： 定数の定義

このように enum キーワードに続けて名前（識別子）、その後”{“と”}”の間に名前（識別子）を並べることによって有限個の値の集まりから成る型を定義できます。これを**列挙型**といいます。列挙では上の例にもあるように名前の後に”{“と”}”で挟んでメンバを定義することができます。一つの列挙型に属する値は全て同じ名前、同じ型のメンバを持つ必要があります。実際、上の例では全ての値が caption という名前で文字列型の値を備えています。

ルート・エリアの定義

ゲーム世界の全てのエリアの基準点となるルート・エリアの定義です。

```
    root = {
        location = "/":loc;
        description = "ルート・エリア";
        scale = 1[km];
        geometry = [=[
<?xml version="1.0" encoding="UTF-8"?>
<geometry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.xgmtk.org/Lore/Point Point.xsd"
    xmlns="http://www.xgmtk.org/Point">
    <point label="0" />
</geometry>
]=];
        environment = {
            height = 0.0[m];
        }:Environment;
    }: Area;
```

図 ２５： エリアが成す木構造のルートとなるエリアの定義

ルート・エリアの location メンバは”/”となります。一応 description メンバには”ルート・エリア”と設定されていますがルート・エリアはシステムの必要上存在はしているだけでプレイヤーから見えることは殆どないでしょう。scale メンバには一応 1[km]を設定していますが、後述のように１点しかないジオメトリを使って定義しているのであまり意味はありません。

geometry メンバには基点となるエリアとして一点しか存在しないジオメトリ・スタイルである Point を使います。Point スタイルのジオメトリには地点が一つしかなくその座標

は”0”（ゼロ）です。そのことは geometry の子要素 point が 1 つでその label 属性が”0”で表現されていることに現れています。Point スタイルジオメトリは短いので別ファイルにせずヒアドキュメント形式の XML 文字列として直接記述しています。ヒアドキュメント形式の XML 文字列については「「とある地方」 ： マップ（１）」(p.20) を参照してください。

形式的に存在しているだけであまり意味がないように見えるルート・エリアですが、一つ重要な役割があります。世界全体にデフォルトの環境を定義するということです。…といってもこのルールの世界は単純なので environment メンバにはただ高度を表す height メンバとジオメトリの回転角度を表す rotation メンバだけが設定されています。高度は海抜 0.0[m]、角度は 0 度（0[degree]）となっています。長さを表す[m]と[km]、角度を表す[degree]は basic.lore で定義されている基本単位です。よりリアルなルールでは明度、温度や風向風速といった気候条件その他が定義されることになるでしょう。

PC フォーム：Actor フォームの拡張

ここではこのルールで利用されるキャラクタに関するフォームとルールを定義します。実際にはシステムが提供するActorフォームを拡張して今定義しているルールに必要な機能を追加します。それには以下のようにformキーワードに続けて新たに定義するフォームの名前（ここではPC)、次いでextendsキーワード、そして基底となるフォームの名前（ここではActor）と書き”{“と”}”の間に追加したいメンバやルールを記述します[14]：

```
    form PC extends Actor{
//＜中略＞
    }
```

図 ２６：PC フォーム

このルールは大変シンプルなので簡単なルールを幾つか追加するに留め、メンバは追加しません。以下この PC に追加するルールについて順番に説明します。

ユーザの行動を尋ねるルール

最初は Actor を介してそれを操るプレイヤーに行動計画を尋ねるルール choice です。

```
        rule choice{
            // リストから選択して貰う
            selected = self.query("どうしますか？",
                {
                    message = "行動を選択してください";
                    action = Action.input;
                });
```

[14] これはオブジェクト指向に親しんでいる方にはお馴染み、クラスの継承です。

```
        // ユーザの選択したアクションで場合分け。
        select(selected.action){
            case(Action.search){
                self.searchAction;
            }
            case(Action.move){
                self.moveAction;
            }
            default{
                self.sleepAction;
            }
        }
    }
```

図 ２７：行動選択問い合わせルール choice

長く見えますが機能は 2 段階からなります。

1 段階目は先ほどアクションのとして定義した列挙 Actions という列挙型からどれかの値を選んでもらうために Actor の（そして PC が継承している）ルール query に問い合わせ用のフォームを渡しその結果を selected という変数に設定することです。

2 段階目は selected の内容に応じて場合分けして Action の詳細を尋ねる 3 つのルールをそれぞれ適用することです。

またこの choice で使われている self キーワードはフォームの中で指定されたルールでだけ使うことができる特別な変数の名前です。self 変数はルール適用時に関連付けられたフォーム・オブジェクトを指します。例えば PC フォームのルール choice を適用する式は PC オブジェクト pc について pc.choice となりますが、呼び出された choice の処理本体では呼び出された際の pc を self という名前でアクセスできるということになります。

以下、具体的に 2 段階の処理について説明します。

まずは 1 段階目。query ルールはそのアクターのプレイヤーのクライアント・ソフトウエアの画面でダイアログを表示し、プレイヤーに何らかの入力や選択をしてもらいその結果を返すというルールです。query の引数の 1 つめはダイアログのタイトルで 2 つめはダイアログの仕様です。ダイアログの仕様は form 名を指定しない無名のフォーム値で指定します。このフォームでは通常の値を渡すとそれはクライアント側でそのままダイアログ内に表示され、型名の後ろに".input"と指定した値になっているメンバについてはその型の値の入力や選択を促すコントロールがダイアログ内に表示されます。そして".input"と指定したメンバが入力、選択された値に置き換わった無名のフォームが結果として返されます。

この例の場合ではダイアログのタイトルは”どうしますか？”となりフォームの message メンバに指定された"行動を選択してください"はダイアログにそのまま表示され、action メンバには Action.input が指定されているためメッセージの下に列挙型 Actions の 3 つの値を選ぶためのドロップダウン・リストが表示されることになります。ユーザがどれかの行動、例えばsleepを選択するとこのqueryルール無名フォーム・オブジェクトを返しselectedの値は以下のようになります：

```
selected = {
    message = "行動を選択してください";
    action = Action.sleep;
};
```

2 段階目は条件分岐でcase(…){…}という節を伴ったselect(…){…}文が使われます。selectの()内の式の値がcase節の()の値にマッチ [15]する最初のcase節が選ばれてその{}の中の処理が実行されます。全てのcaseがマッチしないときはdefault節の{}の中の処理が実行されます。この例ではselected.actionの結果をActionの値と比較し、一致した場合にその選択に合った行動を実現する関数を呼びます。Action.search「探す」ならserchActionルール、Action.move「移動する」ならmoveActionルールが適用されます。それ以外が選択された場合（つまりAction.sleep「寝る」が選択された場合）に実行されるdefault{}の処理はsleepActionルールの適用です。

「探す」アクションを実現するルール

「探す」アクションを実現する searchAction ルールを示します：

```
rule searchAction{
    // キャラクタがアクションしたことをエリアに知らせて
    // トリガをチェックさせる。
    self.action(Action.search);

    /*
    「探す」の 1 ラウンド後に再び予定を尋ねるためにイベント登録
    schedule はシステムで定義されている変数。
```

15 「マッチ」の意味は式の型によって異なります。実際にはその型の値の match というルールが呼ばれ、その結果が真となる場合にマッチしたことになります。多くの場合に等しいと真となりますがそうでない場合もあります。例えば case の値が range 型の値で、select の式の結果が数値型である場合は select の式の値がその range 型の値で表される範囲内に入る時に真となります。

```
        */
        self.futureChoice(1[R]);
    }
```

図 ２８： 「探す」を実現する searchAction 関数

serchAction ルールではまず self.actionTrigger という Actor のルールを適用してシステムにトリガをチェックさせます。ここで渡した Action 型の列挙値 search とマッチする値が action 要素に指定されていて、かつ地理的条件（デフォルトではアクターがトリガと同一地点にいる）を満たすトリガがあればそのトリガが発火することになります。この仕組みによって特定の場所で特定の行為をしたときだけ起こる事象を表現することができます。先のシナリオの例では「とある地方」と「とあるミニマムなダンジョン」という 2 つのエリア間間相互の移動や宝箱の発見に「探す」を利用しています。

ついで次に行動を尋ねるイベントを登録するために PC フォームのルール futureChoice ルールを適用します。このルールでは「探す」は 1 回行うのに 1 ラウンドしか掛からないことにするので futureChoice には 1[R]を引数として指定します。そうすれば 1[R]後に choice ルールが適用されるようにイベントが登録されます。

寝るアクションを実現するルール

「寝る」アクションを実現する sleepAction ルールを示します：

```
    rule sleepAction{
        period =self.query("何時間寝ますか？",
            {
                message = "数字で時間を入力してください。";
                hour = unit[hour].input;
            });
        // 睡眠終了後に再び予定を尋ねるためにイベント登録
        self.futureChoice(period.hour);
    }
```

図 ２９： 「寝る」を実現する sleepAction 関数

sleepAction ルールはは一定時間何もしないことを実現します。そのためまずは先ほど choice ルールでも利用した query ルールを使ってプレイヤーに睡眠時間を 1 時間単位で入力させます。

ついで次に行動を尋ねるイベントを登録するために PC フォームのルール futureChoice ルールを適用します。futureChoice には period.hour を引数として指定します。そうすれば入力された時間が経過したのちに後に choice ルールが適用されるようにイベントが登録されます。

細かい話ですが、futureChoice の引数は[R]単位の入力を求められている一方入力は[hour]単位です。とはいえ[R]と[hour]は共に[sec]を基底として定義され、内部的には秒数で値が保持、計算される同じ単位ファミリ（p.38、図 ２２参照）に属するので[hour]は[R]単位に暗黙の変換をされることになります。ここで[R]は 5 分単位、[hour]は 1 時間単位ですので綺麗に割り切れるため暗黙の変換で丸められても誤差の心配はありません。

移動アクションを実現するルール

アクションを実現する３つのルールの最後は移動を実現するための moveAction ルールです。

```
rule moveAction{
    // キャラクタが現在いるエリアの取得
    area = self.currentArea;

    /*
    プレイヤーに対してキャラクタのいるエリアのマップを表示して
    目的地ととルートを選んでもらいます。
    */
    route = area.choiceRoute(self);

    // 距離と標準的な移動速度から所要時間を計算します。
    period_sec = (route.distance.toReal / standard_velocity.toReal).toInt[sec];

    // ラウンド単位で時間を合わせるため所要時間を
    // ラウンド単位に切り上げます。
    period =unit[R]. ceil(period_sec);

    // 指定された時間経過後に移動完了の処理をするための
    //イベントを登録します。
    self.futureMove(delay, route.goal);
}
```

図 ３０： 「移動する」を実現する moveAction 関数

moveAction ルールはエリア内の移動の際に目的地をユーザに指定して貰い現在位置との間で距離を計り、標準的な移動速度を利用して移動時間を計算します。この移動時間をラウンド単位の移動時間に切り上げて移動時間を計算します。そうして得られた移動時間と目的地を指定して self.futureMove ルールを呼び出して移動を実現します。

移動時間と目的地の計算の詳細は以下のようになります。

アクター（self で示される）が現在いるエリアは self.currentArea で得られ、それを変数 area に設定します。この変数 area に対して pc を引数に area. choiceRoute ルールを適用すると、マップの表示 GUI を通じてユーザに目的地とそこまでの経路を選んでもらうことができます。選んでもらった経路を変数 route に設定します。ここで route 変数の型は Lore 言語があらかじめ用意している Route フォームのオブジェクトです。

さらに変数 route に対して distance ルールを適用すると、現在位置と目的地 2 点間で経路沿いの距離が得られます。この距離を変数 distance に設定します。そして distance を標準的な移動速度（定数 standard_velocity）で割って所要時間を得ます。

また細かい話ですが、Lore 言語の単位型では組立単位を特に識別しないので割り算は一旦双方を基底となる単位の実数値（距離は[m]単位の実数、速度は[m/sec]単位の実数）に変換して計算して[sec]単位の実数値を得ます。これを四捨五入で[sec]単位の整数値へ丸めてから、結果となる[sec]型の値に改めて変換することで行います。この値が period_sec 変数に設定されます。

futureMove ルールの引数は[R]単位なので、period_sec 変数に設定されている[sec]単位の所要時間を unit[R]の ceil ルールに渡すことで[R]単位に切り上げます（ここで意識的に切り上げないで自動変換に任せると四捨五入で丸められてしまうので所要時間が短いと所要ラウンド数が 0 になってしまう可能性があります。）。この値が period 変数に設定されます。最後に所要時間を表す period 変数と目的地のロケーションパスを示す route.goal を futureMoveルールに渡して所要時間経過後に移動を実現するイベントを登録して貰ってこの moveAction ルールの処理は完了です。

ちなみにこのように定義された関数はマップの種類（ジオメトリのスタイル）に依存せずどのような種類のマップでも動作します。

移動イベントを登録するルール

移動イベントを登録するルール futureMove を示します：

```
rule futureMove(period : unit[R], location : loc){
    schedule.add({
        ticks = schedule.current+period;
        priority = 0;
        handler = @{
            self.moved(location);
        };
    }:Event);
    futureChoice(period);
```

```
    }
```

図 ３１： 「一定時間後に移動完了」を実現する futureMove 関数

これは Lore 言語が提供する schdule について、schedule.current ルールによって現在時刻を取得しこれに所要時間 period を足した時刻（優先度は 0）に実行されるイベントを作成し、schedule に登録します。このイベント・オブジェクトの handler メンバにはアクターの位置を書き換えるルール moved に目的地のロケーションパス location を渡して適用する処理が書かれています。イベントの登録には schedule.add ルールに作成した Event オブジェクトを一つ渡して適用します。

イベントはスケジューラで時間順に並べて管理されており、ある時刻、ある優先順位のイベントが終わると同時刻で次の優先順位のイベントが実行され、ある時刻のイベントが全て実行されると、次の時刻のイベントを探し、現在時刻が次のイベントに設定されている時刻へと更新されます。このようにして登録されたイベントが順に実行され、終了する毎に時刻が進んでいきます。

このような仕組みにすることで厳密に時間を管理しつつ、不等間隔に時刻を進めることができます。プレイヤーの実時間で見るとき、ゲーム内の一定時間当たりで登録されているイベントが少ない場合（休息、待機や平穏な長旅など）はゲーム内時間がサクサク進み、登録されているイベントが多い（緊迫したシーン、典型的には戦闘中や隠密行動中など）はゲーム内時間がじりじりと進むことになります。

その後は futureChoice を同じ period を指定して適用することで移動完了時に次の行動選択を尋ねるイベントの登録も行います。

このルールは moveAction ルール以外に、「とある地方」と「とあるミニマムなダンジョン」の間の移動のためのトリガでも用いられます。

移動完了処理を行うルール

移動完了処理を行うルール moved を示します：

```
alter moved(goals：list<loc>){
    // キャラクタの現在位置を設定します。
    self.location ≔ goals.get1st;
}
```

図 ３２： 実際にキャラクタの位置を更新するルール

このルールは呼ばれるとアクター（self）のロケーションを引数 goals に指定されたロケーションパスへと書き換えます。これによってアクターの移動が実現します。

このようにフォーム・オブジェクトの値を更新するルールは特に alter というキーワードで宣言し、書き換える値のリストを受け取る決まりになっています。値の書き換えは通常

の変数の初期化で用いる”=”演算子とはではなく”:=”演算子が利用されます。”:=”演算子は alter で定義されたルール（**更新ルール**）でのみ利用することができます。更新ルールは特殊なルールであり、呼ばれても即座に変更が反映される訳ではありません。以下値の変更が特別な仕組みで行われている理由を説明します。

これまでおおまかに説明してきたある時刻、ある優先度で登録されたイベントの実行は実際には２つのフェイズに分かれており、これまで見てきた Event の handler メンバに登録された処理、そこで適用されたルールはその一つ目のフェイズ（**発火フェイズ**）で実行されています。更新ルールはそこで適用した場合、指定した値とともに適用が予約されるだけで即座には適用されません。実際同時刻、同優先順位で複数のイベントの並行実行が許されており、同じオブジェクトの同じメンバへの書き換え要求は同時に発生し得ます。

そこで２つ目のフェイズ（**更新フェイズ**）では同じ時刻、同じ優先順位の複数のイベントに由来する同一オブジェクト、同一の更新ルールの適用予約が集約され、複数の更新予約で指定された値のリストの形で更新ルールに渡されます。更新ルールはこの複数の更新要求を調停する機能を担っています。このような理由で更新ルールは発火フェイズで引数として単一の値をとって適用予約され、更新フェイズで実際に適用されるときは値のリストを受け取るという特殊な仕様になっています。

これはルールによって、同じルール体系内でも更新しようとする値の種類によって色々な調停の方法があり得ます。例えば一体のアクターが同時に複数の攻撃を受けたダメージであれば単純に合計されることも多いでしょう。移動であればベクトル合成のように加算されるかもしれません（あるアクターが移動中に横から別のアクターに体当たりされるとかの場合)。ルールによっては最大値や最小値を取ることもあるでしょう（体調を表現する列挙型の値の書き換えの場合、最悪のコンディションが選択されるなど)。あるいは新たに調停用のイベント（同時刻でより大きな優先順位値を持つようなイベントを登録できる）を登録し､GM やプレイヤーにダイアログを出して選んでもらうようなルールの実現も可能です。

このような仕様により同時刻同優先順位の複数のイベントの発火フェイズにおいて並行に適用される複数のルールの実行中にどんなメンバの値も変化することはなく、ルールの適用処理が干渉しないことが保証されます。

とはいえこのサンプルのルール体系では同時に複数の位置更新要求が発生しないことがルール体系の設計から保証されているため、更新ルール moved は必ず 1 つの要素だけを持ったリストを受け取ります。従って図　３２の例では単に最初の１個（0 番目）の値を取り出して、それによりアクターの位置（self.location）を更新しています。

次の予定を聞くイベントを登録するルール

次の予定を聞くイベントを登録するルール futureChoice を示します：

```
rule futureChoice(period : unit[R]){
    schedule.add({
        ticks = schedule.current+period;
        priority = 1;
        handler = @{
            self.choice;
        };
    }:Event);
}
```

図 ３３： 次に予定を聞くイベントを登録するルール

これは引数に指定された時間が経過後に実行されるイベントを一つ登録します。そのため ticks は現在時刻と指定された period の和となっています。priority が 1 となっているのは同時刻を指定して優先順位 0 で発行されたイベントの結果が確定してからユーザに選択を尋ねるようにするためです。handler には先に説明した choice ルールを適用する処理が書かれています。今回のルール・サンプルのように全てのアクションが完了する際に必ずこの futureChoice が適用されるようにしておくことでアクターがあるアクションを終えると次のアクションを尋ねられるという繰り返しが実現されます。

Actor フォームを拡張した PC フォームの定義は以上で終わりです。

シナリオ初期化処理

開始直後に発火する scenario.lore の first イベントで適用されているシナリオ初期化のルールです。

```
rule initial{
    system.allPC.foreach(@(actor : Actor){
        select(actor){
            as(pc : PC){
                pc.choice;
            }
            default{
                //Do nothing, ignore.
            }
        }
    });
```

```
    }
```

図 ３４： 初期化関数

そしてこの initial ルールはシステムが自動的に収集しているシナリオ内の全てのアクター（いゆる NPC も含みます）のリスト system.allPC に対し、foreach ルールを使って各アクターの行動予定を尋ねるために choice 関数を呼び出します。

foreachルールはリストが備えるルールで、リスト内の各要素に対して指定された処理を実行します[16]。foreachに指定された処理内のみからアクセスできるactor変数には処理の対象となる各Actorオブジェクトが 1 つ設定されています。

とはいえここには風変わりな処理が書かれています。それは as(…){…}節を伴う select(…){…}文です。これは choice ルールが Actor フォームを拡張した PC フォームで定義されている一方、system.allPC というリストは Actor のリストなので型が合わないためこのような書き方が必要になるのです。

Actor フォームを拡張して PC フォームを作った今回の例の場合、PC オブジェクトは Actor オブジェクトが備えるメンバやルールは全て備えているので特に何ともなく Actor オブジェクトとして安全に扱うことができます。従って Actor オブジェクトのリストに Actor オブジェクトやそれを拡張したその他のフォームのオブジェクトと混在して PC オブジェクトが格納されることには何の問題もありません。しかし逆に Actor オブジェクトとしか分かっていない値を PC オブジェクトとして扱うことについては、拡張により Actor フォームに存在しないメンバやルールが PC フォームには追加されているので安全ではありません。

そこでas(…){…}節を伴うselect(…){…}文を使ってActorオブジェクトが実際にPCオブジェクトであるかを確認してそれが正しい場合だけ処理を行うようにしているのです[17]。具体的にはselectの()内の式（ここではactor）が実際にas()内の型宣言（ここではpc：PC）で示されるPCフォームである場合にのみas節の{}の中の処理が実行されます。その際pcはselectの()内の式と同じフォーム値を指していますが、そのフォームはPCになっているので変数pcを介して安全にPCで追加されたメンバやルールを利用できます。Actorを拡張して複数のフォームを作ることができるのでas(){}は必要なだけ書くことができます。default{}節はas(){}節に合致するものがない場合に実行されます。この例ではPCオブジェクト以外のActorオブジェクトはいないのでdefault節では何もしていません。

---

[16] これは実際には並列に実行されます。即ちこの例ではアクターとそのプレイヤーが複数いる場合、同時に全員に問い合わせが行き、全員が応答を終えるとこの処理は終了します。

[17] Java プログラマな読者の方はお察しの通りこれは if(actor instanceof PC){PC pc = (PC)actor;/*…pc を使った処理…*/}という構文と同じく型の判別とダウンキャストの機能を実現するものです。

PCTrigger ： Actor の拡張に対応する Trigger の拡張

ルールの最後はあまり面白くないちょっとしたフォームの拡張です。

```
form PCTrigger extends Trigger{
    handler = @(actor : Actor){
        select(actor){
            as(pc : PC){
                pcHandler(pc);
            }
            default{
                //Do nothing, ignore.
            }
        }
    };

    pcHandler : @<(PC)>;
}
```

図 ３５： PCTrigger フォーム

シナリオで利用した PCTrigger では発火すると実行される処理を保持するメンバ pcHandler はアクターとして PC オブジェクトを受け取る仕様になっていました。しかし Lore 言語があらかじめ用意した Trigger は発火すると handler の処理を実行し、その際には Actor オブジェクトを受け取る仕様になっています。両者を整合させるには前節で説明した select…as 構文で変換する必要があります。このため拡張により新しく pcHandler という処理を保持するメンバを新設してそれを呼び出す処理を handler へ記述します。メンバ pcHandler に指定された@<(PC)>という型名は PC オブジェクトを受け取る処理を表します。

以上のようにして幾つかの単位、定数、列挙、フォーム、ルールを記述することで「移動する」「探す」と「寝る」しかできないミニマムなルールが記述できます。このミニマムなルールを先のシナリオを合わせることで実際に実行可能なシナリオとなります。

2.4 状態の保存

Lore 言語はルール、シナリオの記述の他に進行中（あるいは終了した）のセッションの状態の保存の記述フォーマットとしても利用する予定です。scenario.lore が実行され途中で 3 度保存されたという想定で作成したサンプルが図 ３６です：

```
docinfo{encoding="UTF-8",version="http://xgmtk.org/lore/1.0":url}
```

```
desc("[saved session] サンプルシナリオ")
    {“Lore言語のサンプルコード、シナリオの記述例”}
author("椎路 ちひろ"){"Chihiro_Shiiji@xmpp.xgmtk.org":jid,
    "mail:develop@xgmtk.org":url}
history("2013-07-15T00:00:00+09:00":date){reviser="[GM]椎路 ちひろ",
    desc="セッション保存"}
history("2013-11-05T00:00:00+09:00":date){reviser="[GM]椎路 ちひろ",
    desc="セッション保存"}
history("2013-12-22T00:00:00+09:00":date){reviser="[GM]椎路 ちひろ",
    desc="セッション保存"}

// シナリオのインポート
import "basic.lore":url;
import "rules.lore":url;
import "scenario.lore":url;

/*
シナリオ開始後に保存されるファイルには_auto_storedセクションが作られ、
シナリオ進行中に増えたオブジェクトが記録される。
この部分のフォーマットは暫定的な物で、開発が進むにつれて変わるであろう。
*/
section _auto_stored {
    // GMのJIDを記録。
    master = “chihiro@xmpp.xgmtk.org”:jid;
    // 典型的にはセッション内の現在時刻及び、
    // イベント・オブジェクトが以下の形式で保管される。
    schedule = {
        ticks = 86400;//シナリオ開始からゲーム時間で1日後
        priority = 0;

        events={
            {//登録されたイベントの例
                ticks = 86700;//シナリオ開始からゲーム時間で1日+1R(5分）後
                priority = 0;
                handler = @{
                    scenario.pc.moved("/region/dungeon[1,2]":loc);
```

```
                };
            }:Event,
            {//登録されたイベントの例
                ticks = 87000;//シナリオ開始からゲーム時間で1日+2R(5分）後
                priority = 0;
                handler = @{
                    scenario.pc.choice;
                };
            }:Event
            /*
            発行されたイベントが処理済も含めて全て記録されるので実際はもっと多いが、
            これはサンプルなので省略。
            */
        };
    }:Schedule;
}
```

図 ３６： HelloWorldSaved.lore

セッションの保存情報は今後開発が進むにつれて変わると予想されますが、ここではごく簡単にどんな情報が保存されるかについて述べます。

１つめは history 文で履歴の記録です。Chihiro@xmpp.xgmtk.org という JabberID を持つユーザが 2012-08-07T02:07:30.00+09:00 という日時（ISO 8601 形式）に保存を行ったことを示しています。これは保存の時に自動で追加されるであろうものです。履歴は更新や保存の度に自動あるいは手動で追記されていき history 文の数が増えていきます。

２つめは_auto_stored セクションの master 変数で GM が Chihiro@xmpp.xgmtk.org という JabberID を持つユーザであることを記録しています。

３つめは現在時刻とシナリオ中から登録されたイベントのリストを記録する _auto_stored セクション schedule 変数です。

この他セッション中に動的に追加されたオブジェクトは_auto_stored セクションに保存されていくことになるでしょう。このように Lore ML 言語を利用してシナリオの実行によって変化した状況も記述、保存することができます。

３まとめ

以上、本稿ではまず TRPG のルールとシナリオを記述する Lore 言語を幾つかのサンプルを例に概説しました。

Lore 言語は XGMTK のようなルール主導型オンライン・セッション・システムの制御の根幹をなしており、またデータを共有する共通フォーマットでもあります。ルールを記述する上での Lore 言語の特徴はイベントによる離散的な時間管理に基づくシミュレーション・プログラミング言語であることです。しかし、シナリオ記述のレベルではその複雑さは隠ぺいされており、エリア、アクター、アイテム、トリガ、シークレットといったオブジェクトを配置するだけで簡単にシナリオを記述することができます。そして TRPG のルール・シナリオの記述を容易にするため、範囲、単位、ダイス、ロケーション、エリアといった各種のデータ型を備えています。

本稿で紹介したサンプルの Lore ファイル、及びそれを検証した際のパーサは GitHub 上で公開しています：

https://github.com/TakayukiKando/Lore

４ 今後のスケジュール

Lore 言語は XGMTK というプロジェクトの一環として開発しています。XGMTK は図 ３７に示すように、TRPG を対象に特定のルールに依存しない汎用のオンライン・セッション・システム（インフラとクライアント）、及びシナリオ&ルールのオーサリング・ツールの開発を目指すプロジェクトです。

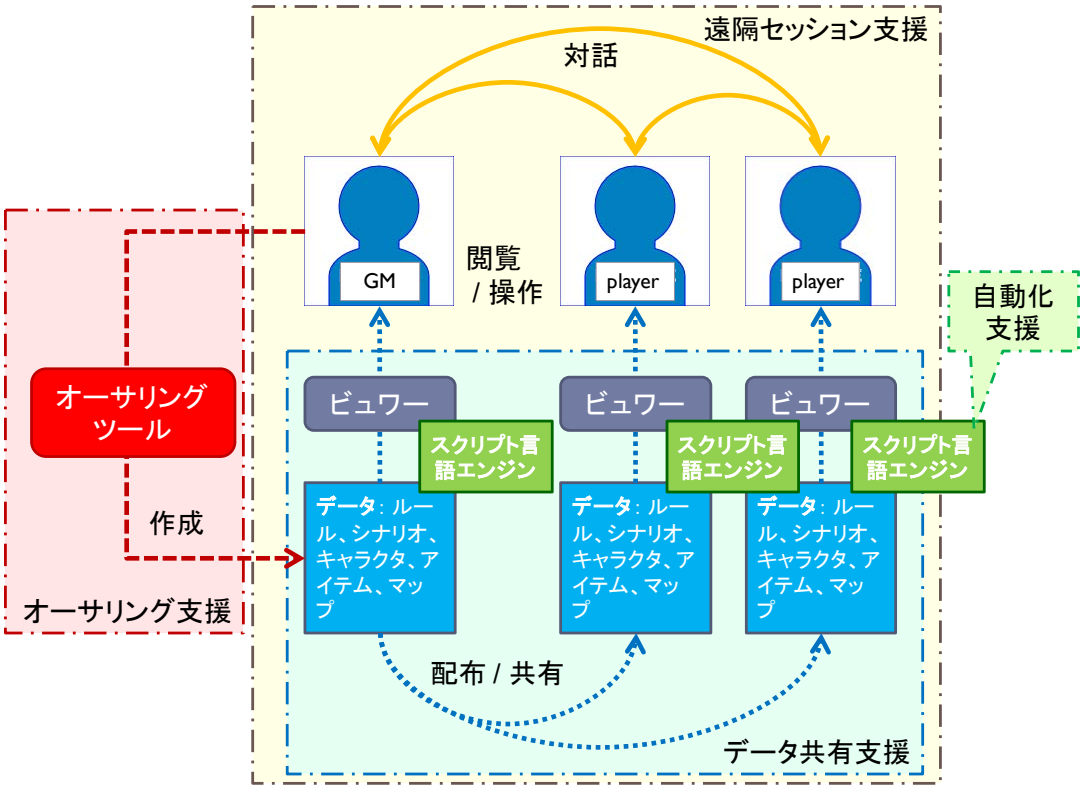

図 ３７：XGMTK プロジェクト

Lore 言語はこの XGMTK プロジェクトで TRPG に必要なルールやシナリオを記述し、交換する共通フォーマットを目指して設計開発されています。現在 ANTLR 4 用の文法記述が完了して Lore 言語パーサは動いています。実際にルール主導型セッションシステムを駆動するにはこのパーサを使ってインタプリタを実現する必要があります。

今後は夏コミまでに Lore 言語インタプリタを稼働させ、サンプルの例題として RuneQuest のルールを記述して、Lore 言語や基本ライブラリの仕様を改良していく予定です。

協力者は随時募集しております。特に GUI 詳しい人とか、スマホ、タブレット詳しい人とか。あと本をカッコ良くレイアウトできる人とか！（←ダサい自覚はあるらしいw）

あとがき

今回、言語の文法の再定義に伴い No.3 の内容を改訂しました。現在 ANTLR 4 で文法を定義して解析木のビジタ＆リスナの Java コードを生成してテスト実行してみたところで力尽きています。なんとか夏までにインタプリタが動くようにしたいところです。そこまでいけばなんとかデモが実行できるようになるので。「わかりにくい」というコメントを頂いている XGMTK と Lore 言語がどういうものかについてもイメージし易くなるのではないかと思っています。

（↑は 3 人セットで書いた涼宮ハルヒ・シリーズの TFEI の皆さんから朝倉さん。）

2013 年 12 月 28 日

椎路 ちひろ

XGMTK report No.6

2013 年 12 月 31 日発行
編集・発行：神戸隆行
（〒819-0015 福岡市西区愛宕 3-2-28-301）
develop@xgmtk.org
http://www.xgmtk.org/
GitHub: https://github.com/TakayukiKando/Lore
Twitter: @ChihiroShiiji