

モデリングツール astah\* を使って

# 演習テキスト

アーキテクチャを活かした開発

これは冗談ぽいけど  
なにか表紙があるといいね



株式会社チェンジビジョン 編・著



モデルを使ってソフトウェアを開発しよう

astah\* を使った UMLチュートリアル

株式会社チェンジビジョン

バージョン pdf\_0011, 2022-01-17

# 目次

はじめに .....	1
この文書の目的 .....	1
対象・目標 .....	1
アイコンの説明 .....	1
注意事項 .....	3
諸注意 .....	4
商標等について .....	4
1 準備 .....	5
1.1 モデリングツール( astah* )を用意する .....	6
1.2 実装環境を用意する .....	6
1.3 演習用のプロジェクトを用意する .....	7
1.4 ボウリングのスコアのつけ方 .....	9
2 解決したい課題はなにか .....	12
2.1 手動のときの手順を整理する .....	12
2.2 自動化されてからの手順を整理する .....	24
2.3 ユースケースを吟味する .....	25
2.4 解決したいのはゲームの進行とスコアの計算 .....	40
3 スコアの構造を図にする .....	41
3.1 ゲームスコアのオブジェクト図を描く .....	42
3.2 ゲームスコアのクラス図を描く .....	55
4 スコア記録の振る舞いを図にする .....	64
4.1 振る舞いの図を選択する .....	64
4.2 ステートマシン図を描く .....	65
5 まとめ .....	80
6 他の図、他の機能など .....	81
付録A: 関連資料、ワークシート等 .....	83
用語集 .....	84
参考文献 .....	86

# はじめに

## この文書の目的

この文書は、ソフトウェアの開発にモデリングツールを使う方法などについて独習するためのチュートリアルです。

このチュートリアルは、ソフトウェアの開発に、UMLを使って描いたモデルが役に立つことを実感する機会を提供することを目的としています。主に、これからソフトウェア開発にモデル図を使うことについて学ぶみなさんを対象にしています。また、モデリングツールを使うことにも慣れてもらえるよう、「astah\* Professional」を使って演習します。

## 対象・目標

### チュートリアルの対象者

このチュートリアルは、次のような方を対象者と想定しています。

#### チュートリアルの対象者

- ・ モデリングやUMLについてこれから学ぼうとしているみなさん
- ・ ソフトウェア開発工程は知っているが、まだモデル図を活用していないみなさん
- ・ 分析・設計・テストなどソフトウェアの開発に携わっているみなさん

### チュートリアルの目標

みなさんがチュートリアル終了後に次のような人になっていることを、このチュートリアルの目標としています。

#### チュートリアルの目標(人物像)

- ・ 開発工程のどこでどのモデル図を使えばよいか知っている(知っている人になる)
- ・ 「開発にはモデルがあったほうがいい」と考えている
- ・ 自分だけでなく他の人にもモデリングを勧めたいと考えている

## アイコンの説明

本文中では、次のようなアイコンを用いています。



| 告知。みんなが覚えておくとよい追加情報など。



ティップス。覚えておくと便利なちょっとしたヒントやコツ。



重要。間違えたり見落とすと期待通りの結果が得られない設定や操作など。



注意。気をつけないと問題の発生につながるようなことがら。



警告。守らないと破損や怪我などにつながる可能性のあることがら。



# 注意事項

## 諸注意

- ・本文書は、株式会社チェンジビジョン(作成者)が編集したもので、本文書に関する権利、責任は作成者が保有します。
- ・本分書に記載されている情報は、本文書を更新した時点のものであり、利用時にはURLなどの各種の情報が変更されている可能性があります。
- ・本文書は演習用テキストとしての使用を想定し、内容等は予告なしに変更することがあります。また、作成者がその内容を保証するものではありません。
- ・本文書の内容に誤りや不正確な記述がある場合も、作成者は一切の責任を負いません。
- ・本文書に記載の内容や実施結果からいかなる損害が生じても、作成者は責任を負いかねますので、あらかじめご了承ください。
- ・本文書の複製、保存については、作成者の同意を得てください。

## 商標等について

- ・Windows 並びに同社の各製品名は米国マイクロソフトコーポレーションの米国およびその他の国における登録商標です。
- ・Apple、iCloud、iPad、iPhone、Mac、Macintosh、macOSは、米国およびその他の国々で登録されたApple Inc.の商標です。
- ・Linuxは Linus Torvalds氏、米国及びその他の国における登録商標あるいは商標です。
- ・UNIXは、X/Open Company Limitedが独占的にライセンスしている米国ならびに他の国における登録商標です。
- ・その他、本書に記載されている社名、製品名、ブランド名、システム名などは、一般に商標または登録商標でそれぞれ帰属者の所有物です。
- ・本文中では ©、®、™、は表示していません。



ここで、チェンジビジョンの保有する商標等について言及する。



# 1 準備

このチュートリアルでは、モデリングツールとして astah\* を、実装にはRubyを使います。最初に、演習に必要な環境を用意しましょう。

## 1.1 モデリングツール( astah\* )を用意する

このチュートリアルで使用するモデリングツールを用意しましょう。

astah\* の入手や評価版を含むライセンスの取得方法などについては、次のページを参照してください。

astah\* の入手や取得方法に関するページを用意する。

astah\* の取得方法

[URL](#)

astah\* のライセンスの入手方法



[URL](#)

astah\* のインストール

[URL](#)

astah\* のライセンスの追加方法

[URL](#)

## 1.2 実装環境を用意する

このチュートリアルでは、実装に Ruby を使います。執筆時点では 2.7.2 を使いました。

Ruby は、Windows、Mac、Linuxで動作するオープンソースの言語プログラミング言語です。インストール方法は利用環境によって異なりますので、詳しくは、Rubyの公式サイトの説明やWebの記事を参考にしてください。

### Rubyのインストール

<https://www.ruby-lang.org/ja/documentation/installation/>

# 1.3 演習用のプロジェクトを用意する

業務やシステムを分析(あるいは設計)するときは、たいてい複数種類のモデル図を使って表します。astah\* では、複数のモデル図をまとめてたモデルファイルを「プロジェクト」と呼んでいます。



astah\* のプロジェクトの作り方についての詳しい説明は、次のページを参考にしてください。

astah\* 機能ガイド プロジェクトの作成と利用

<https://astah.change-vision.com/ja/manual/278-project-file.html>

この演習で使うプロジェクトを用意しましょう。新規プロジェクトを作成したら、先に名前をつけて保存しておくとよいでしょう。

## 演習で使うプロジェクトを作成する

1. astah\* を起動する。
2. 「ファイル」メニューから「プロジェクトの新規作成」を選択する( 図 1.1 )。

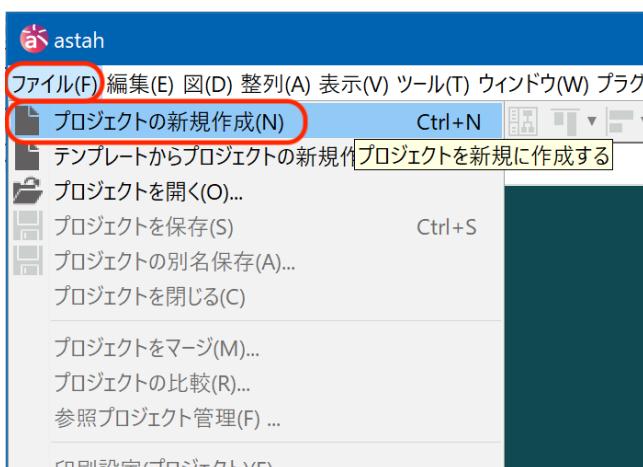


図 1.1 新しいプロジェクトを作成する

3. 「no\_title」という名前で新しいプロジェクトが作成される( 図 1.2 )。

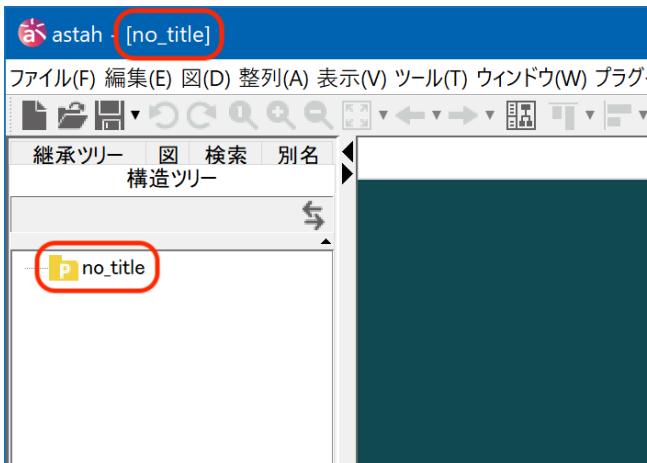


図 1.2 新しいプロジェクトが作成された

4. 「ファイル」メニューから「プロジェクトを保存」を選択する( 図 1.3 )。

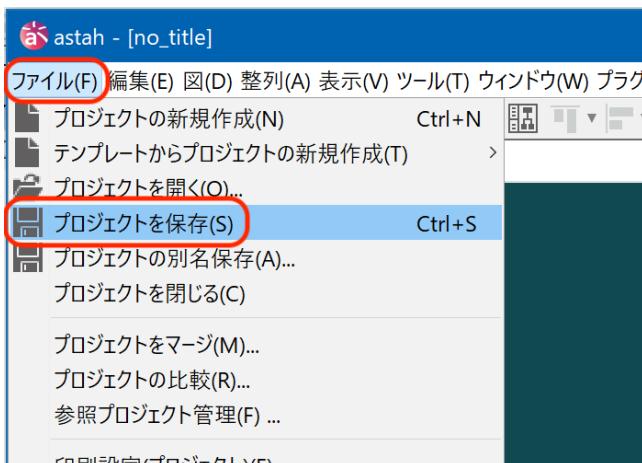


図 1.3 新しいプロジェクトを保存する

5. 「保存」ダイアログが開くので、プロジェクトの保存場所と保存するファイル名を指定して、「保存」ボタンをクリックする。ここでは保存場所として「デスクトップ」に「BowlingScore」フォルダを作成し、「bowling\_score」を指定した( 図 1.4 )。

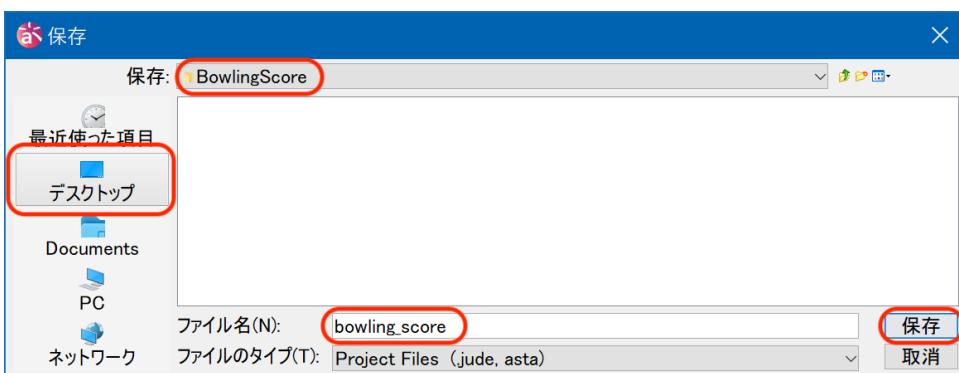


図 1.4 ダイアログを操作してプロジェクトを保存する場所とプロジェクトファイル名を指定する

6. 保存すると、プロジェクトファイルにつけた名前が「構造ツリー」に反映される( 図 1.5 )。

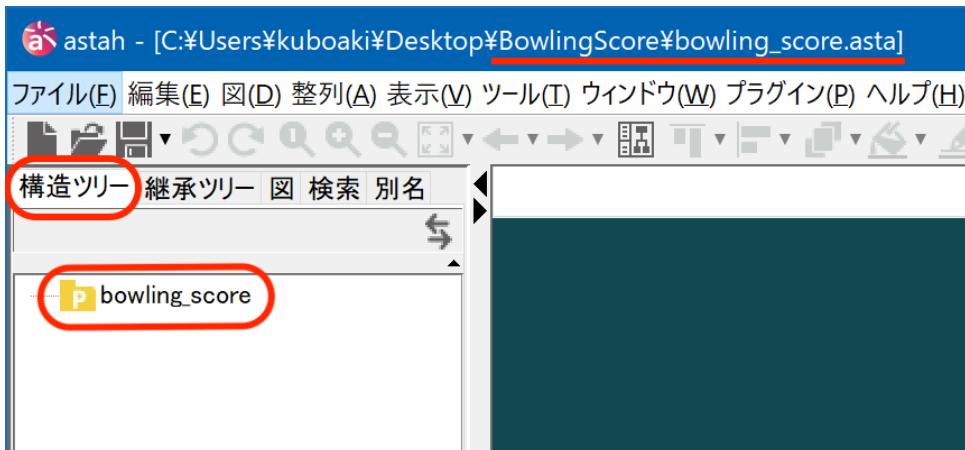


図 1.5 ダイアログを操作してプロジェクトを保存する場所とプロジェクトファイル名を指定する

## 1.4 ボウリングのスコアのつけ方

ボウリング場では、だいぶ前から自動でスコアが表示されるようになっています。自動化される前は、受付でスコアシートをもらって、自分たちでスコアを計算し、手でスコアをつけていました( 図 1.6 )。



図 1.6 手でスコアを記入している様子

ボウリングの規則とスコアのつけ方について、公益財団法人「全日本ボウリング協会」の「ボウリング競技規則」を参考に、まとめてみました。

### 公益財団法人「全日本ボウリング協会」のボウリング競技規則

<https://jbc-iwate.com/service/message/201012101.pdf>

#### 1.4.1 クラシックスコアリング

従来から使われていて、広く普及しているスコア計算方式です。

##### ゲームの構成

- ・ボウリングの1ゲームは、10 個のフレームをもって構成する。
- ・競技者は、ストライクの場合を除き、それぞれのフレームで 2 回ずつ投球する。ただし、第 10 フレームがストライク又はスペアの場合には、サービスフレームとして、ストライクの場合は、さらに 2 回投球でき、スペアの場合

には 1 回投球できる。

- ・ ゲームの成績は、適正な投球によって、倒されたピンの数をもって計算し、10 個のフレームの合計点によって、これを表す。
- ・ 適正に投球されたボールとは、競技者を持っているボールが、手から放れファールラインを越えたものをいう。

表 1.1 ボウリングスコアに使う記号と意味

記号	名称	意味
	ストライク	1投目で10本全てを倒した場合。ストライクをだしたフレームの得点は、次の2投分を加算する。2回続いたら3フレーム目の1投目までを加算する。スコア欄は、それまでは空欄にしておく。ストライクが続くことを「ダブル」「トリプル(ターキー)」と呼ぶ。
	スペア	1投目で残ったピンを、2投目で全部倒した場合。次の1投分を加算する。スコア欄は、それまでは空欄にしておく。
	ミス	フレームの2投目でピンを1本も倒せなかった場合。2投目でガターに落ちても「G」の記号は使わずにこの記号を使う。そのフレームの得点は、1投目で倒したピン数になる。なお、1つのフレームで2回投球し、10本のピンを全部倒すことができなかった場合をエラーという。
<b>G</b>	ガター	1投目でレーンの両脇にある溝(ガター)にボールが落ちた場合。2投目はガターに落ちた場合も「ミス」として扱う。
	スプリット	1投目で1番ピン(ヘッドピンともいう)と他のいくつかのピンが倒れ、2本以上のピンが次のような状態に残った場合。スプリットの場合は、1投目のピン数を○で囲む。  * 残っているピンの中間のピンが少なくとも1本倒れたとき。例えば7と9あるいは3と10。 * 残っているピンのすぐ前のピンが少なくとも1本倒れたとき。例えば5と6。
<b>F</b>	ファール	ファールラインを越えて投球されたという記号。1投目でファールした場合、ピンを倒しても1投目のピン数は0になる。2投目は全ピンを立て直す。2投目でファールした場合、ピンを倒しても2投目のピン数は0になる。第10フレームの3投目がファールの場合も、ピンを倒しても3投目のピン数は0になる。

名前	1	2	3	4	5	6	7	8	9	10	Total	
くぼあき	6 9	3 18	9 —	G 21	3 38	8 58	7 78	9 96	8 104	F 130	6 150	150/150

図 1.7 スコアの例

## 1.4.2 カレントフレームスコアリング

2018年アジア競技大会で採用された新しいスコア計算方式です。

### クラシックスコアリングと異なる点

- 1投目によって10ピンすべてを倒した場合はストライクとなる。ストライクのとき、そのフレームの得点は30となる。
- 続けて2回ストライクの場合は、それぞれのストライクの得点は30となる。
- 1投目の後、残ったピンを2投目によって全部倒した場合は、スペアとなる。スペアのとき、そのフレームの得点は、1投目のピン数に10を加えた数となる。
- 第10フレームでストライクあるいはスペアをとった場合も、3投目はない。

名前	1	2	3	4	5	6	7	8	9	10	Total
うえはら				(8) 	F 8	9 F		8 —	G 9		202/202

図 1.8 スコアの例(カレントフレームスコアリングの場合)

### 【豆知識】ボウリングの名前の由来

「ボウリングをすること」などを意味する「bowl」という英語は、ラテン語で「泡」や「瘤」を意味する「bulla」に由来する。一方、同じ綴りで食器や容器(ボウル)を意味する「bowl」や「球」を意味する「ball」は、ゲルマン語に由来し、本質的に異なる。

— Wikipedia

## 2 解決したい課題はなにか

スコアをつけるときの解決したい課題はなんでしょうか。スコアをつけるときの手順を整理しながら、考えてみましょう。

### 2.1 手動のときの手順を整理する

スコアを手でつけていたときは、スコアを計算してスコアシートに記録するのは、スコアラーの仕事です。スコアラーになった人は、ゲームの様子を観察して倒れたピン数を確認し、スコアをつけるルールに従ってスコアをつけます。仲間うちでやるときは、投球する順番がきているプレーヤーがボールを投げている間、手の空いている他のプレーヤーが交代でスコアラーになってスコアをつけます。

このときのプレーヤーとスコアラーの関係を「ユースケース図」で表してみましょう。ユースケース図は、関心の対象となっている業務やシステムが提供したい機能やサービスと、それらを利用する人や外部のサービス等の関係を表すために使います。



機能やサービスを提供する主体のことを、ユースケース図では「システム」と呼ぶことが多いです。そのサービスが人によって実施される作業を含んでいても、それらのサービスを提供する範囲をシステムと呼びます。

#### 2.1.1 プロジェクトに分析モデルを追加する

まず、プロジェクトに分析モデルを追加しましょう。ここで言う分析モデルは、分析用に作成するモデル図を格納する場所だと思ってください。



ここでの分析とは、対象とする業務やサービスを詳しく調べて、役割を持つ複数の要素に分けることだと考えておけばよいでしょう。

##### プロジェクトに分析モデルを追加する

1. プロジェクトにモデルを追加する。
  - 構造ツリー上で、プロジェクトを選択する。
  - 右クリックしてポップアップメニューを開き、「モデルの追加>モデル」を選択する( 図 2.1 )。

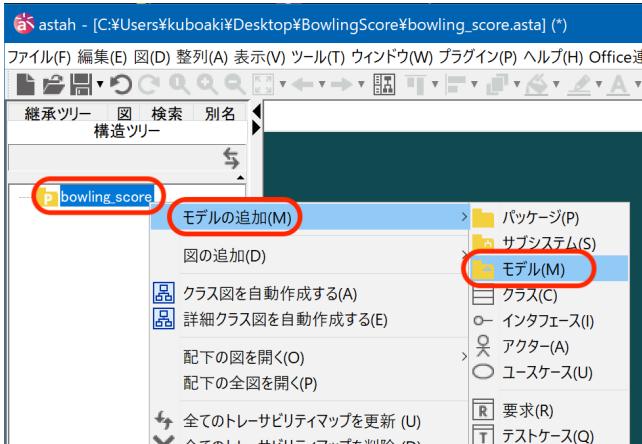


図 2.1 プロジェクトにモデルを追加した

## 2. 追加したモデルに名前をつける

- 構造ツリー上で、追加したモデルを選択した状態で、プロパティの「ベース」タブを選択する。
- 「名前」を編集して「01\_分析モデル」とする(図 2.2)。

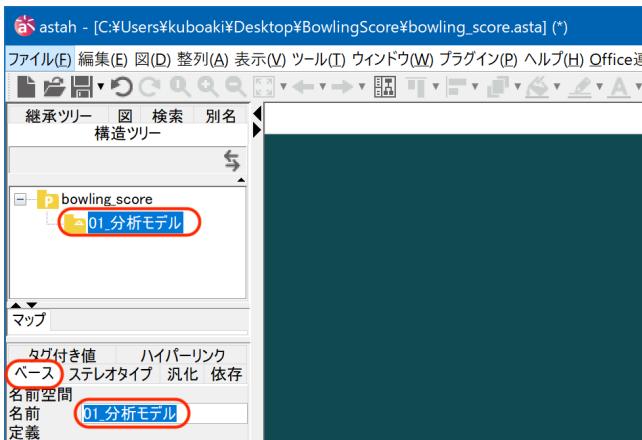


図 2.2 モデルに分析用のモデルとして名前をつけた



モデル名の前に「01\_」とつけたのは、今後作成するモデルを自分の考える順に並ばせるための工夫です。

## 2.1.2 分析モデルにユースケース図を追加する

次に、分析モデルにユースケース図を追加して、手動でスコアをついているときのユースケース図を描いてみましょう。

### モデルにユースケース図を追加する

- 構造ツリー上で、「01\_分析モデル」を選択する。
- 右クリックしてポップアップメニューを開き、「図の追加>ユースケース図」を選択する(図 2.3)。

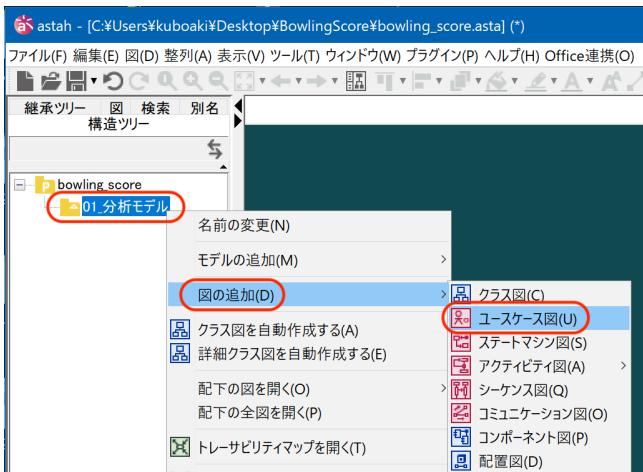


図 2.3 モデルにユースケース図を追加する

## 3. 追加した図に名前をつける。

- 構造ツリー上で、追加したユースケース図を選択した状態で、プロパティの「ベース」タブを選択する。
- 「名前」を編集して「手動でスコアをついているときのユースケース図」とする(図 2.4)。

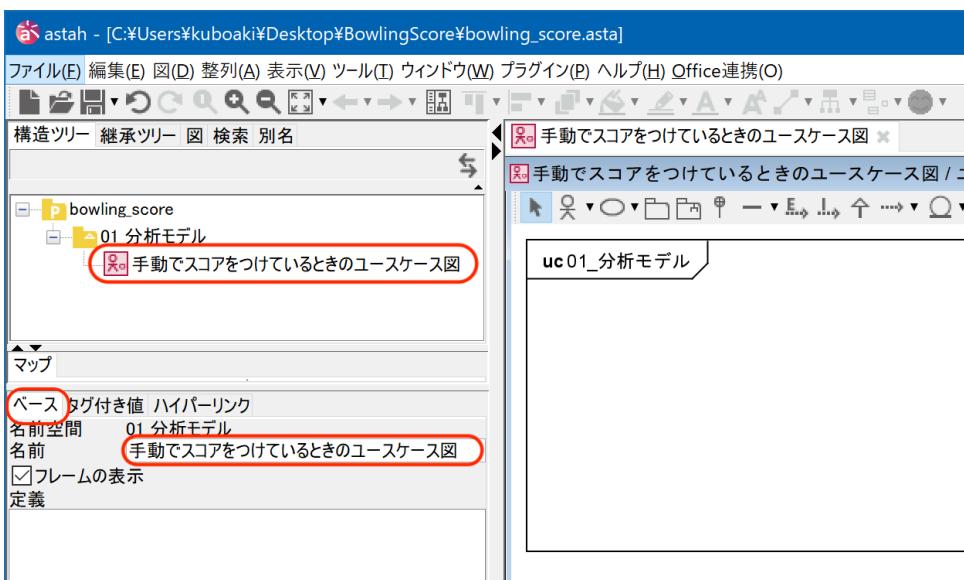


図 2.4 プロジェクトにユースケース図が追加できた

これで、プロジェクトにユースケース図が追加できました。

## 2.1.3 ユースケース図にアクターを追加する

ユースケース図が追加できましたので、こんどはユースケース図にアクターとユースケースを追加しましょう。

ユースケース図では、システムが提供する機能やサービスを「ユースケース」呼びます。そのユースケースを利用する、外部の人や外部のサービスのことを「アクター」と呼びます。



アクターは人とは限りません。外部の機器や、サービスを提供する別のシステムの場合もあります。

まず、アクターについて考えてみましょう。スコアラーにスコアを計算して記録してもらっているのはプレーヤーですね。つまり、このシステムからサービスを享受しているアクターはプレーヤーだとみなせそうです。

追加したユースケース図にアクター「プレーヤー」を追加しましょう。

### アクター「プレーヤー」を追加する

1. パレットからアクターのシンボル(人型のアイコン)を選択する( 図 2.5 )。

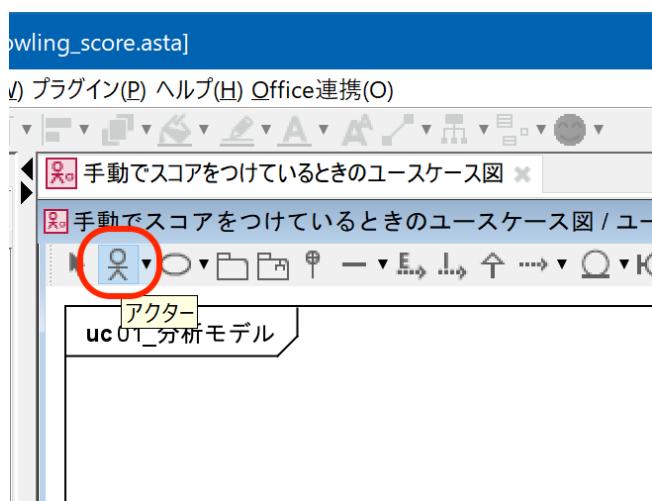


図 2.5 パレットからアクターを選択する

2. 図の中でマウスをクリックすると、アクターが追加される( 図 2.6 )。

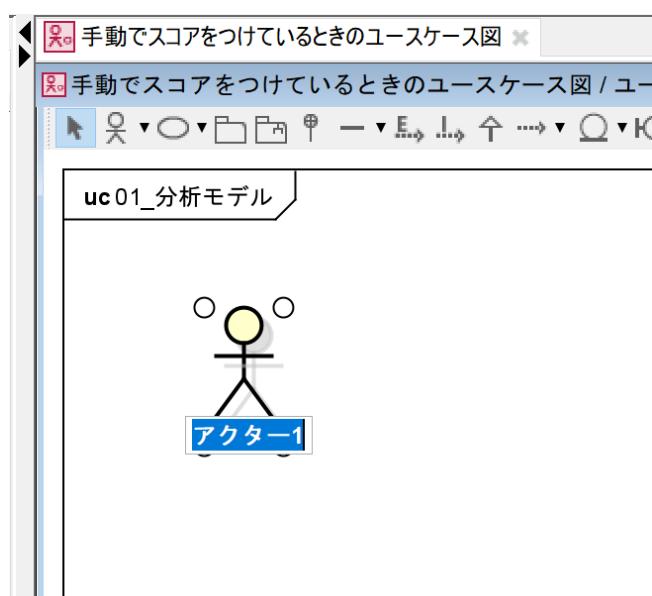


図 2.6 アクターが追加された

3. アクターの名前を編集する。

- アクターの名前が選択されている状態(図2.7)で「プレーヤー」に変更する。
- または、アクターが選択されている状態で、プロパティの「名前」欄を編集する(図2.8)。

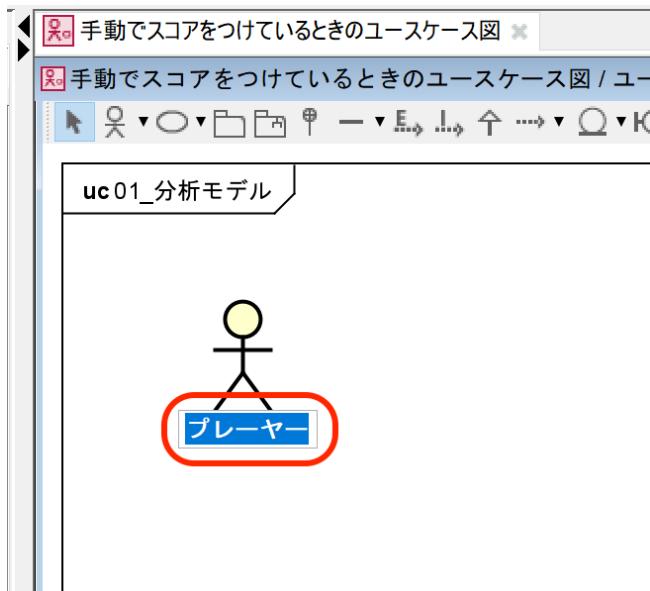


図 2.7 アクターの名前を編集する(直接編集)

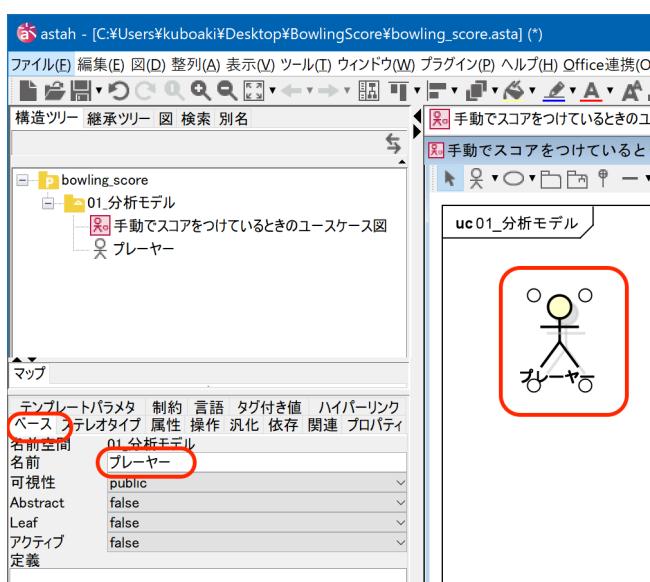


図 2.8 アクターの名前を編集する(プロパティから編集)

これで、ユースケース図にアクター「プレーヤー」が追加できました。



あるシステムに対して、アクターはひとつとは限らず、複数見つかる場合もあります。また、同じ利用者であっても、いくつかの役割や立場からシステムを利用する場合もあります。アクターは、役割ごとに用意し、役割が分かれる名前をつけるようにしましょう。

## 2.1.4 ユースケース図にユースケースを追加する

次に、ユースケースについて考えてみましょう。いま分析しているのは、手動でスコアをつけているスコアラーの仕事でしたね。つまり、システムとして捉えようとしているのは、スコアラーが担当する仕事というわけです。これは、このユースケース図そのものが表そうとしていることでもあります。

このような捉え方をした上で、スコアラー(システム)がプレーヤー(アクター)に提供しているサービスにあたるものを考えます。すると、「プレーヤーが倒したピン数を取得して、スコアを計算し、スコアシートに記録する」という一連の処理がこれにあたりますね。この一連の処理に「スコアをつける」という名前をつけましょう。これを「ユースケース」と呼びます。

追加したユースケース図にユースケース「スコアをつける」を追加しましょう。

### ユースケース「スコアをつける」を追加する

1. パレットからユースケースのシンボル(楕円のアイコン)を選択する( 図 2.9 )。

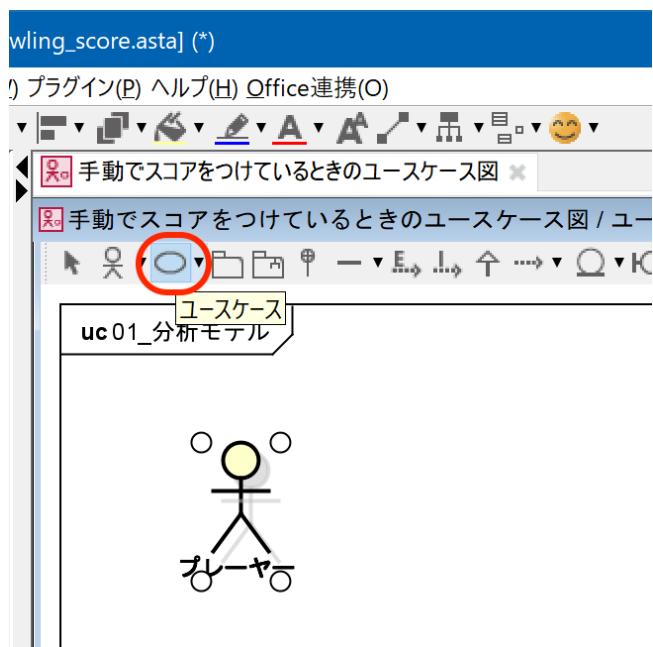


図 2.9 パレットからユースケースを選択する

2. 図の中でマスをクリックすると、ユースケースが追加される( 図 2.10 )。

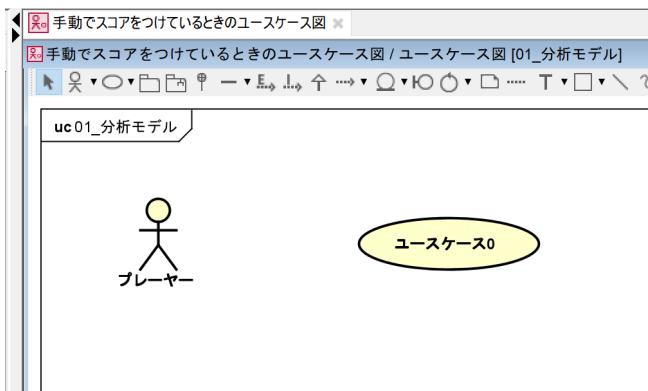


図 2.10 ユースケースが追加された

3. ユースケースの名前を編集する。

- ユースケースの名前が選択されている状態( 図 2.11 )で「スコアをつける」に変更する。
- または、ユースケースが選択されている状態で、プロパティの「名前」欄を編集する( 図 2.12 )。

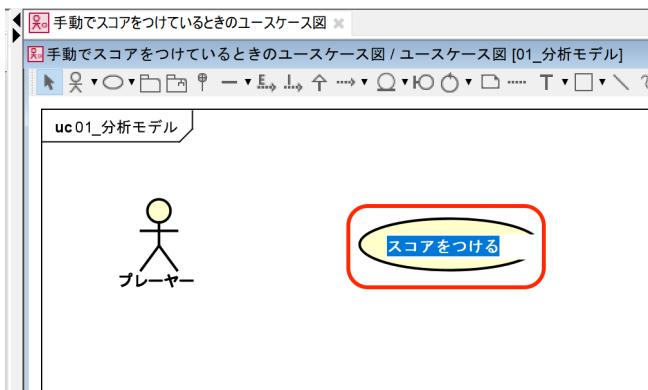


図 2.11 ユースケースの名前を編集する(直接編集)

The screenshot shows the astah tool interface. On the left, the '構造ツリー' (Structure Tree) pane shows a project structure with 'bowling\_score' and '01\_分析モデル' containing a use case named 'スコアをつける'. On the right, the main workspace shows the UML Use Case Diagram with the same use case highlighted with a red border. The properties panel on the bottom left shows the '名前' (Name) field for the selected use case is set to 'スコアをつける', also highlighted with a red border.

図 2.12 ユースケースの名前を編集する(プロパティから編集)

これで、ユースケース図にユースケース「スコアをつける」が追加できました。

## 2.1.5 アクターとユースケースを関連づける

これまでのところ、アクターとユースケースがひとつずつですので、これらが互いに関連しているとみなすのは容易です。しかし、複数のアクターや複数のユースケースがあると、どのアクターがどのユースケースを利用するのかわなくなってしまうでしょう。そこで、アクターと、そのアクターが利用するユースケースの間に関連の線を引いて、ユースケースとアクターを関連づけます。

アクター「プレーヤー」とユースケース「スコアをつける」の間に関連を引いてみましょう。

### アクター「プレーヤー」とユースケース「スコアをつける」を関連づける

1. パレットから関連のシンボル(横線のアイコン)を選択する( 図 2.13 )。

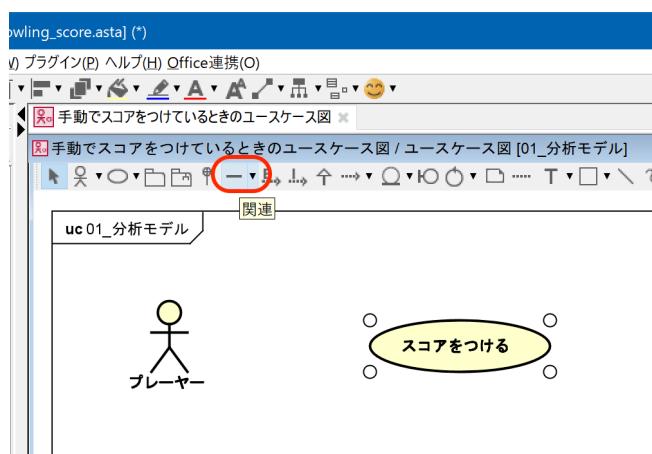


図 2.13 パレットから関連を選択する

2. アクター「プレーヤー」の内部(周辺ではなくシンボル上)でマウスをクリックする。
  - 青い枠線が現れるのを待つ。
  - 枠線が現れたら、マウスをドラッグすると赤い線が現れる( 図 2.14 )。

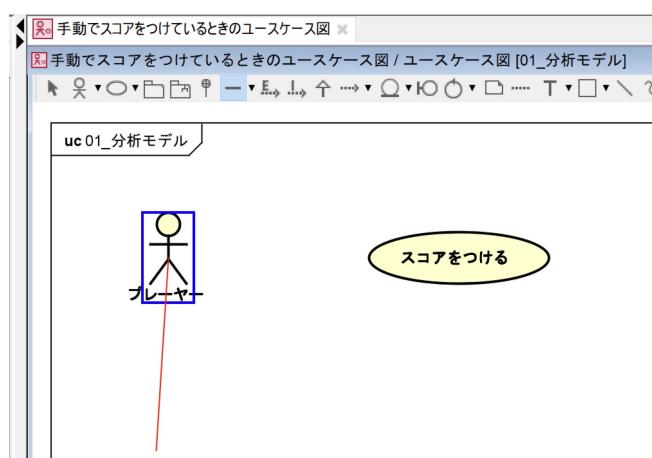


図 2.14 関連を引き始める

3. 赤い線を引いたまま、マウスをユースケース「スコアをつける」の内部まで移動する。

- 青い枠線が現れるのを待つ( 図 2.15 )。
- マウスをクリックすると関連が引かれる。

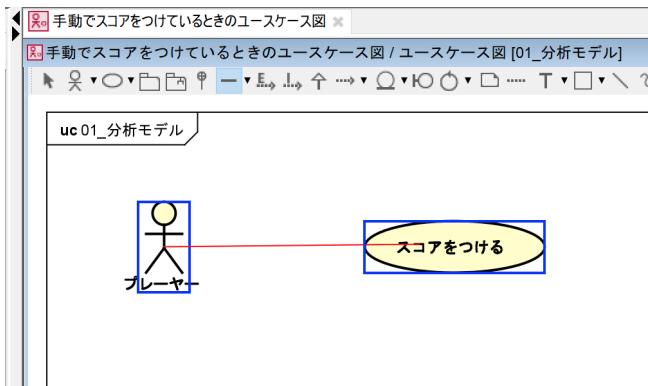


図 2.15 関連が引かれた

これで、アクター「プレーヤー」とユースケース「スコアをつける」の間に関連が引けました。

このユースケース図は、システムとして表しているのはスコアラーの仕事で、プレーヤーに対してひとつのサービス「スコアをつける」を提供していると読みます。

## 2.1.6 ユースケース記述を追加する

ユースケース図の上では、ユースケースやアクターはそれらの名前しか記載しません。そのため、ユースケースの詳しい内容(どのような仕事をするのか)はよくわかりません。そこで、ひとつのユースケースに対して、そのユースケースの一連の処理の流れを説明した記述をひとつ書きます。これを「ユースケース記述」と呼びます。

ユースケース「スコアをつける」のユースケース記述を考えてみました。スコアの計算はスコアラーの頭の中でやります。スコアを記録する作業もスコアラーの手作業です。そうすると、プレーヤーとスコアラーの間でのやりとりは、ピン数のやりとりだけですので、ユースケース記述は「リスト1」のようになるでしょう。

### リスト 1. ユースケース「スコアをつける」のユースケース記述

1. プレーヤーは、投球する。
2. プレーヤーは、スコアラーに倒したピン数を申告する。
3. スコアラーは、スコアを計算して、スコアシートに記録する。
4. ゲームの終了まで、1から3を繰り返す。

astah\* には、ユースケース記述を記述するエディタが用意してあります。

ユースケース記述エディタを使って「リスト1」のユースケース記述を書いてみましょう。

### ユースケース記述エディタを使ってユースケース記述を書く

## 1. ユースケース記述エディタを開く。

- ユースケース「スコアをつける」をマウスを右クリックして、ポップアップメニューを表示する。
- 「ユースケース記述を開く」を選択する。

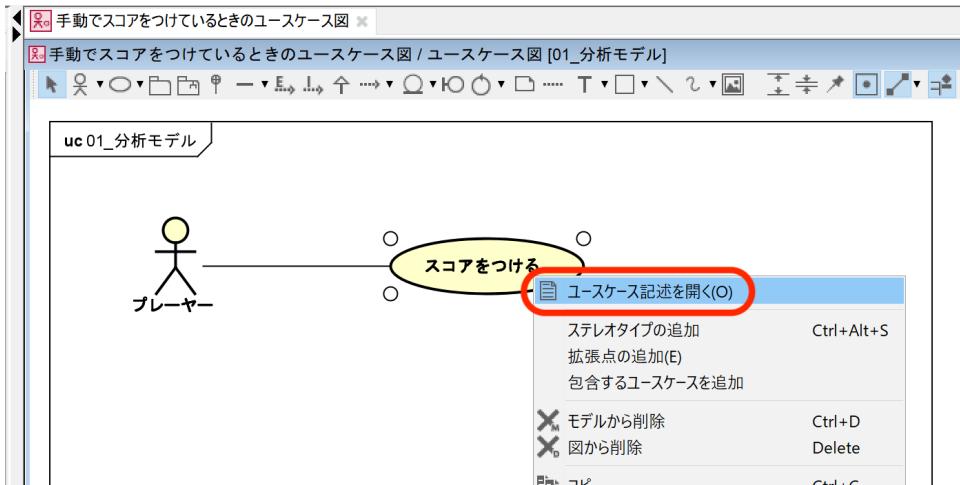


図 2.16 ユースケース記述エディタを開く

## 1. ユースケース記述を書く。

- 「基本系列」に「リスト1」に記載した内容を書き込む

手動でスコアをつけるときのユースケース図 / ユースケース記述 [01_分析モデル]	
項目	内容
ユースケース	スコアをつける
概要	
アクター	プレーヤー
事前条件	
事後条件	
基本系列	1)プレーヤーは、投球する。 2)プレーヤーは、スコアラーに倒したピン数を申告する。 3)スコアラーは、スコアを計算して、スコアシートに記録する。 4)ゲームの終了まで、1から3を繰り返す。
代替系列	
例外系列	
サブユースケース	
備考	

図 2.17 ユースケース「スコアをつける」のユースケース記述



「ユースケース記述エディタ」の記述項目には、基本系列のほかに、代替系列、事前条件などの記述項目があります。これらの項目のうち、この演習では基本系列だけ使っています。他の項目の使い方を学ぶのは、もう少し慣れてからでよいでしょう。

簡便に記述する方法として、「ユースケース記述エディタ」を使う代わりに、「ノート」を使って書く方法を紹介しておきます。

ノートは、UMLの図を作成するときに、補助的な説明をつけるために用意されている要素で、耳を折った矩形で表し

ます。

ノートは、モデルの要素に対して、捕捉する説明をつけたいときに使います。ここでは、ユースケース「スコアをつける」は、どのような手順なのかを説明しています。

ノートを追加して、「リスト1」のユースケース記述を書いてみましょう。

### ノートを使ってユースケース記述を書く

1. パレットからノートのシンボル(耳が折れた矩形のアイコン)を選択する( 図 2.18 )。

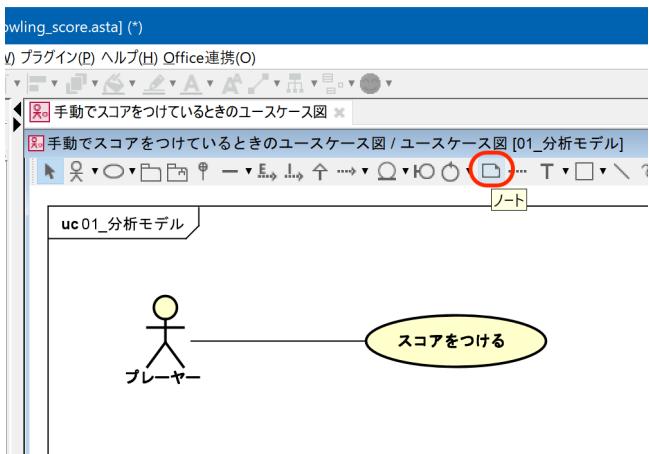


図 2.18 パレットからノートを選択する

2. 図の中でマスをクリックすると、ノートが追加される( 図 2.19 )。

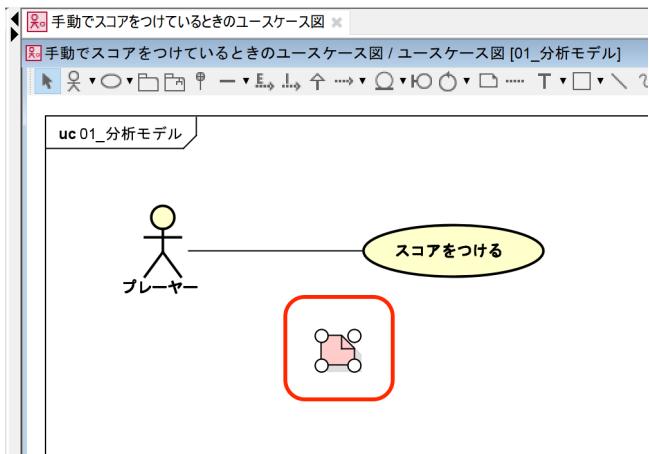


図 2.19 ノートが追加された

3. ユースケース記述を編集する。

- ノートが選択されている状態( 図 2.19 )で、プロパティを編集して、ユースケース記述を書く( 図 2.20 )。

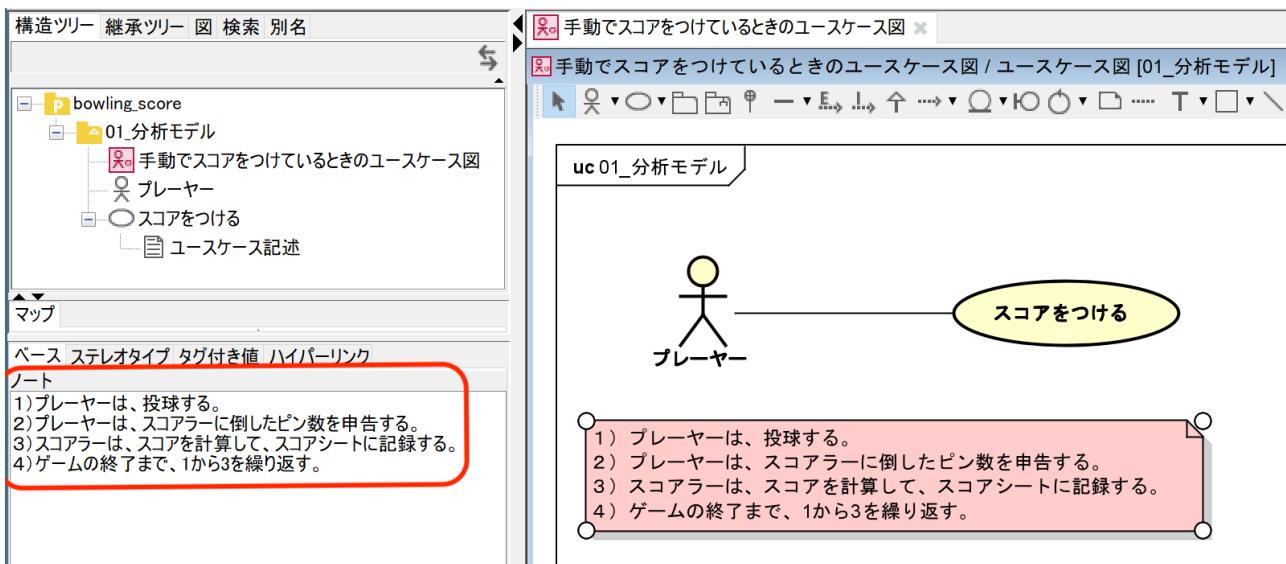


図 2.20 ノートにユースケース記述を書く

4. ノートからユースケースへアンカーを引く( 図 2.21 )。
- パレットから、ノートからユースケースへのアンカーを選択する。
  - ノートの内部へマウスを移動すると、ノートに青枠が現れる。
  - マウスのボタンをクリックし、マウスをユースケースの内部へ移動する。
  - ユースケースに青枠が現れたら、マウスクリックすると、アンカーが引かれる。

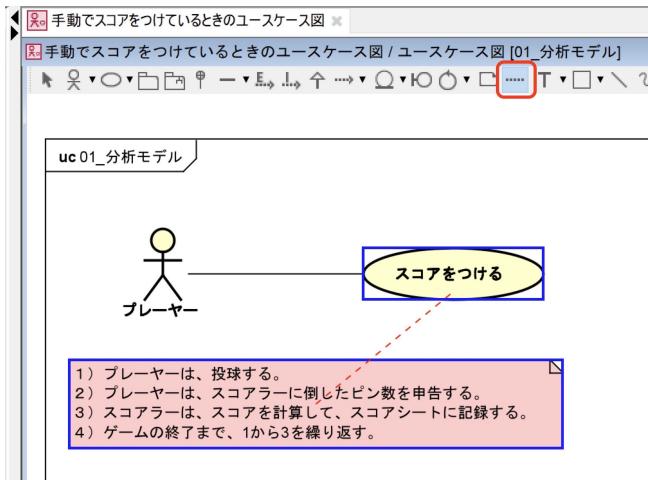


図 2.21 ノートからユースケースへアンカーを引く

できあがったユースケースは「図 2.22」のようになるでしょう。

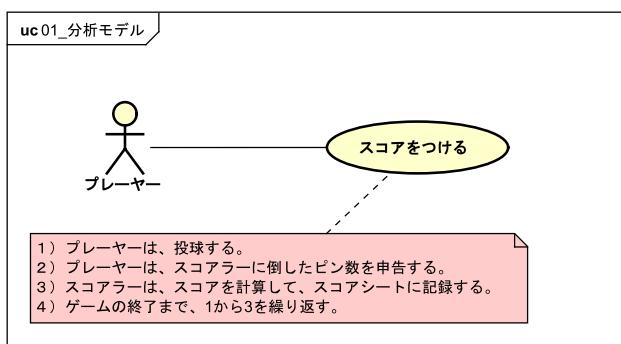


図 2.22 手動でスコアをついているときの(スコアラーの働きを表している)ユースケース図

この図は、「スコアラー(システム)は、プレーヤー(アクター)にスコアをつけるというサービスを提供する」と読めます。そして、「スコアをつける」の詳細はノートを使ったユースケース記述に記載しています。



一般に、ユースケースやアクターはひとつだけとは限らないことに注意しましょう。これまでの分析結果において、アクター「スコアラー」とユースケースは「スコアをつける」だけが見つかっているので、「図 2.22」のような図になっているのです。

## 2.2 自動化されてからの手順を整理する

「図 2.22」は、手でスコアをついている場合を表していました。このままでは、人がやっている仕事をそのまま表しているだけですね。こんどは、スコアラーがスコアをつける手順に着目して、スコアをつける作業の一部を自動化して助けてくれるサービスを想定してみましょう。つまり、こんどは、スコアラーに提供するサービスを考えることになります。

そうすると、「図 2.22」とはユースケース図の構成が変わってきます。スコアラーに提供するサービスを考えてみるわけですから、スコアラーがアクターになりますね。

このように捉えた場合のユースケース図を描いてみましょう。

「モデルにユースケース図を追加する」と同じ手順で、「01\_分析モデル」の中にユースケース図を追加します。図の名前は「スコアをつけるところを助けてくれるサービスを考えたユースケース図」としておきましょう( 図 2.23 )。

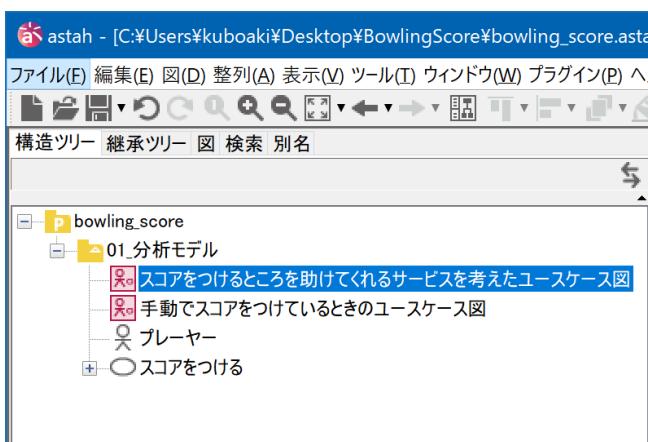


図 2.23 分析モデルにユースケース図を追加する

まず、アクターが「スコアラー」なのは明らかですので、スコアラーを追加しましょう（これはもうできますね）。

次に、スコアラーに提供するサービスを表すユースケースを考えてみましょう。顧客から依頼されている場合であれば、顧客から収集した現状の業務や改善の要求から導出するところです。ここでは、スコアラーがスコアをつけるときに、スコアを計算したり記録する部分を助けてもらいたいのでした。これをユースケースとするのがよさそうです。

ユースケース名は「スコアを（スコアラーに代わって）記録する」にしましょう。

ユースケース記述は、「リスト2」のようになるでしょう。

#### リスト2. ユースケース「スコアを記録する」のユースケース記述

1. スコアラーは、プレーヤーが投球するたびに、ピン数をシステムに入力する。
2. システムは、スコアシートの状況に応じて、ピン数を記録する。
3. 前のフレームのストライクやスペアのボーナスを計算する。
4. ゲームが終わるまで、1から3を繰り返す。

作成したユースケース図は「図2.24」のようになるでしょう。

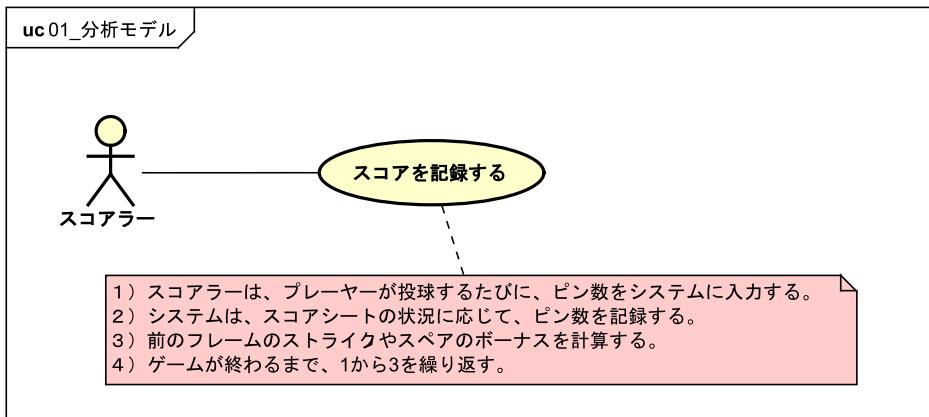


図2.24 スコアをつけるところを助けてくれるサービスを考えたユースケース図

以上の分析の結果、このチュートリアルで作成するシステムは、「スコアラーが使う」「スコアを記録する」システムということがわかりました。

## 2.3 ユースケースを吟味する

こんどは、ユースケース「スコアを記録する」について、もう少し詳しく検討してみましょう。

スコアを記録するときの、スコアラーとシステムの間のやり取りや、入力されたピン数をシステムがどのように扱うのかを検討するために「ロバストネス図」を使ってみましょう。

## 口バストネス分析と口バストネス図

口バストネス図は、システムの構成要素を分析する「口バストネス分析」で使用する図です。口バストネス分析では、システムを構成する要素は「バウンダリ」「エンティティ」「コントロール」の3種類に大別できると考え、該当する要素がないか想定して構成要素を探します。また、これらの要素間の繋がりには制約があると考えることで、構成要素を探しやすくなります。

口バストネス図で抽出した要素は、クラスの候補になります。ただし、ひとつの要素がそのままひとつのクラスになるとは限りません。複数の要素を組み合わせてクラスに仕立てるといったこともあります。

口バストネス分析と口バストネス図の詳しい説明は、「ワークブック形式で学ぶUMLオブジェクトモデリング [robustness01]」や「ユースケース駆動開発実践ガイド [robustness02]」などを参照してください。

### 2.3.1 プロジェクトに口バストネス図を追加する

まず、アクター「スコアラー」とシステムの間のやりとりを検討するための口バストネス図を作成しましょう。そこで、プロジェクトに口バストネス図を追加します。分析の続きですので、「01\_分析モデル」に追加しましょう。口バストネス図はクラス図の表記を応用した図なので、astah\*で作成するときはクラス図を使います。

#### 分析モデルに口バストネス図を追加する

1. モデルにクラス図を追加する。
  - 構造ツリー上で、「01\_分析モデル」を選択する。
  - 右クリックしてポップアップメニューを開き、「図の追加>クラス図」を選択する。( 図 2.25 )。

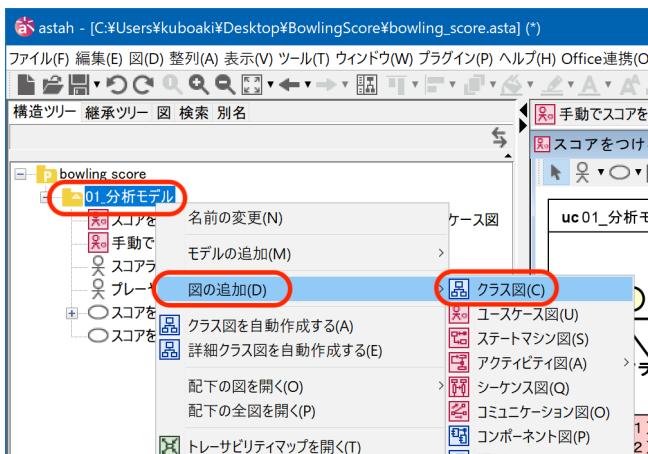


図 2.25 モデルにクラス図を追加する

2. 追加した図に名前をつける。
  - 構造ツリー上で、追加したクラス図を選択した状態で、プロパティの「ベース」タブを選択する。
  - 「名前」を編集して「スコアラーとシステムの間の口バストネス図」とする( 図 2.26 )。

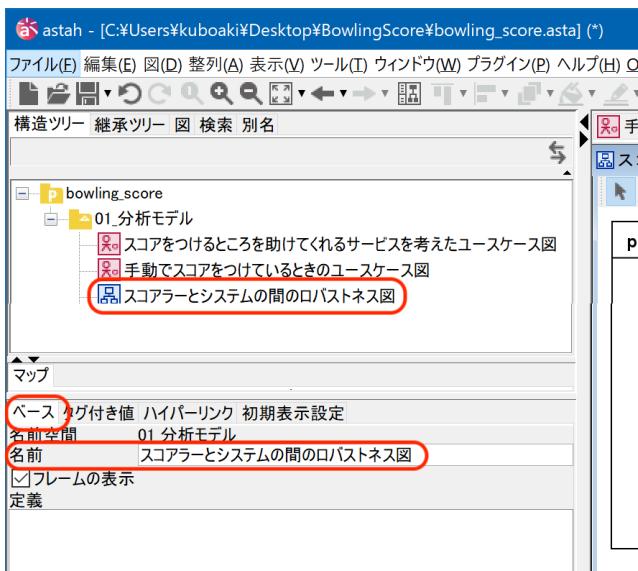


図 2.26 モデルにロバストネス図が追加できた

これで、ロバストネス図が描けるようになりました。

## 2.3.2 新しいゲームを用意する

新しいゲームを始めるとき、スコアラーが手で書いていたときは、スコアシートを用意してゲームに参加する人の名前を書くといった準備をします。

スコアを記録する部分を自動化した場合、新しいゲームを用意するためにはどのような要素が必要になるでしょうか。ロバストネス図を使って検討してみましょう。

まず、アクター「スコアラー」を追加しましょう。

### ロバストネス図にアクターを追加する

- 構造ツリー上から追加したロバストネス図をダブルクリックして、編集できる状態にします( 図 2.27 )。

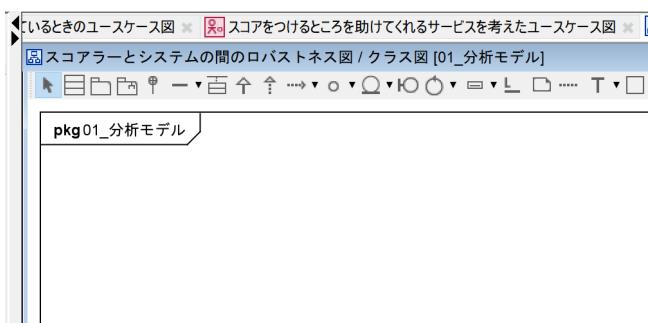


図 2.27 追加したロバストネス図を編集可能にする

- スコアラーは、作成済みで構造ツリー上にありますので、これをドラッグ&ドロップして追加します( 図 2.28 )。

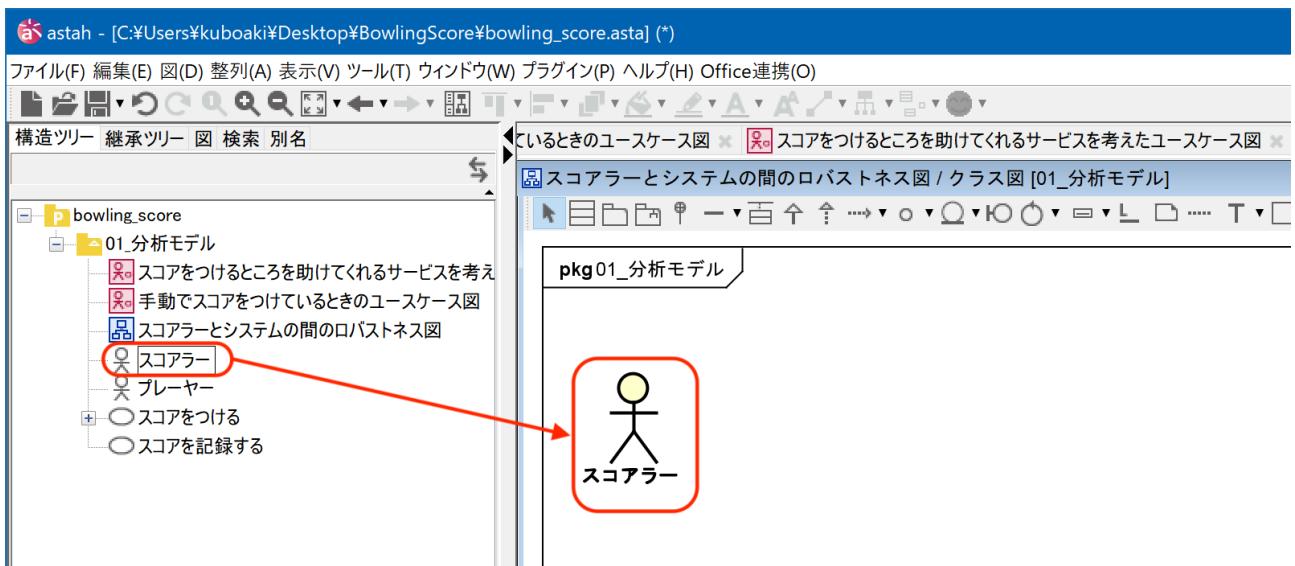


図 2.28 追加したロバストネス図を編集可能にする

次に、スコアラーが新しいゲームを用意するときにシステムとやりとりする場合に必要な要素を、ロバストネス図のシンボルを使って表してみます。

新しいゲームを始めるときは、スコアラーがシステムに新しいゲームを用意する指示を出します。このことを実現するには、システムには、指示を受け取るためのインターフェースと新しいゲームを用意する処理が必要になります。

これらをロバストネス図に追加してみましょう。

#### 新しいゲームを用意するための要素を追加する

1. バウンダリ「ゲーム開始指示入力」を追加する( 図 2.29 )。
  - ・パレットからバウンダリのシンボルを選択し、ロバストネス図に追加する。
  - ・追加したバウンダリの名前を「ゲーム開始指示入力」とする。
  - ・アクター「スコアラー」とバウンダリ「ゲーム開始指示入力」の間に関連を引く。

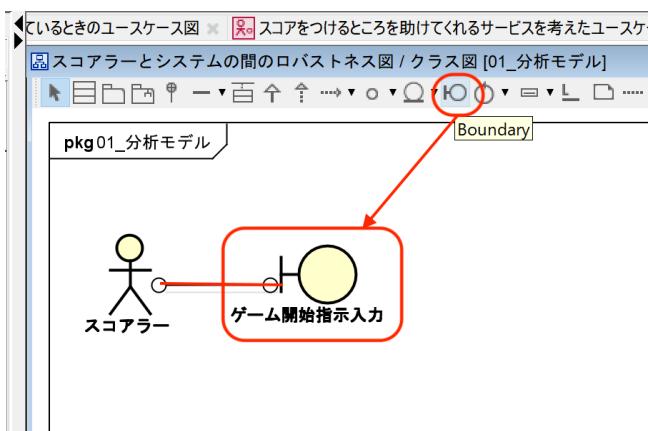


図 2.29 バウンダリ「ゲーム開始指示入力」を追加する

2. コントロール「新しいゲームを用意する」を追加する( 図 2.30 )。
  - ・パレットからコントロールのシンボルを選択し、ロバストネス図に追加する。

- 追加したコントロールの名前を「新しいゲームを用意する」とする。
- バウンダリ「ゲーム開始指示入力」とコントロール「新しいゲームを用意する」の間に関連を引く。

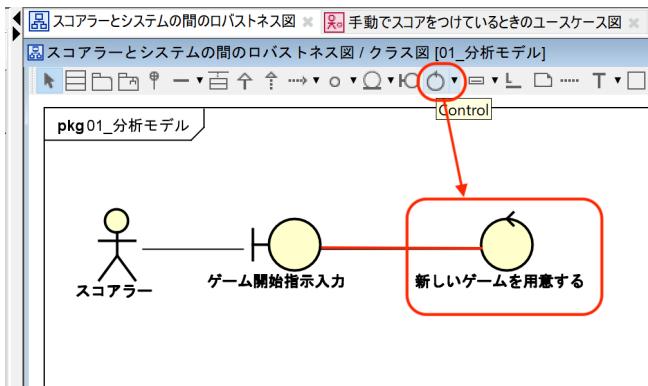


図 2.30 コントロール「新しいゲームを用意する」を追加する

### 3. エンティティ「ゲームスコア」を追加する( 図 2.31 )。

- パレットからエンティティのシンボルを選択し、ロバストネス図に追加する。
- 追加したエンティティの名前を「ゲームスコア」とする。
- コントロール「新しいゲームを用意する」とエンティティ「ゲームスコア」の間に関連を引く。

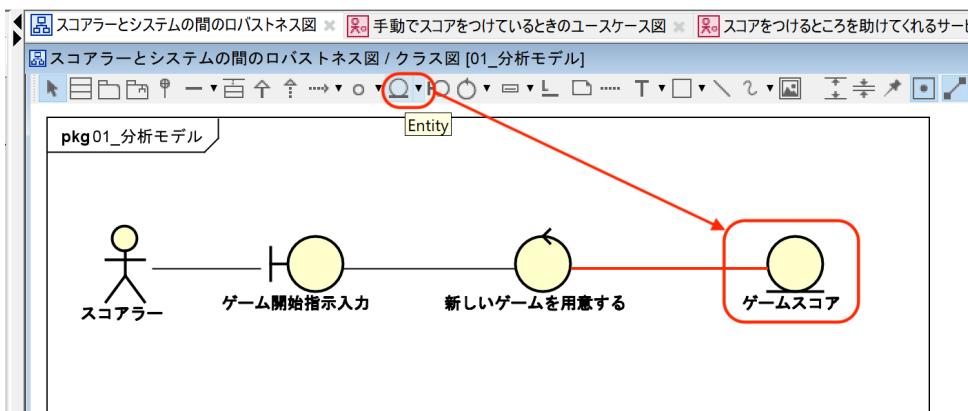


図 2.31 エンティティ「ゲームスコア」を追加する

これで、スコアラーが新しいゲームを始める指示を出したとき、システムが新しいゲームを用意するのに必要な要素が洗い出せました。

## 2.3.3 ゲームを進める

こんどは、ゲームを進めているときに必要な要素を検討してみましょう。

スコアラーがスコアシートを手で書いていたときは、スコアラーはいまどのフレームの何投目であるかを意識して書いていたでしょう。一方、スコアの記録を自動化した場合は、どのフレームの何投目なのかはシステムに把握しておいてほしいでしょう。そうしておけば、スコアラーは単にピン数を入力すれば済むようになります。

そして、システムは、ピン数を取得するたびにゲームを進めます。ピン数を取得するたびに、ゲームスコアを更新し、更

新したスコアをスコアラーへ返せばよさそうです。

これらをロバストネス図に追加してみましょう。

### ゲームを進めるための要素を追加する

#### 1. バウンダリ「ピン数入力」を追加する( 図 2.32 )。

- パレットからバウンダリのシンボルを選択し、ロバストネス図に追加する。
- 追加したバウンダリの名前を「ピン数入力」とする。
- アクター「スコアラー」とバウンダリ「ピン数入力」の間に関連を引く。

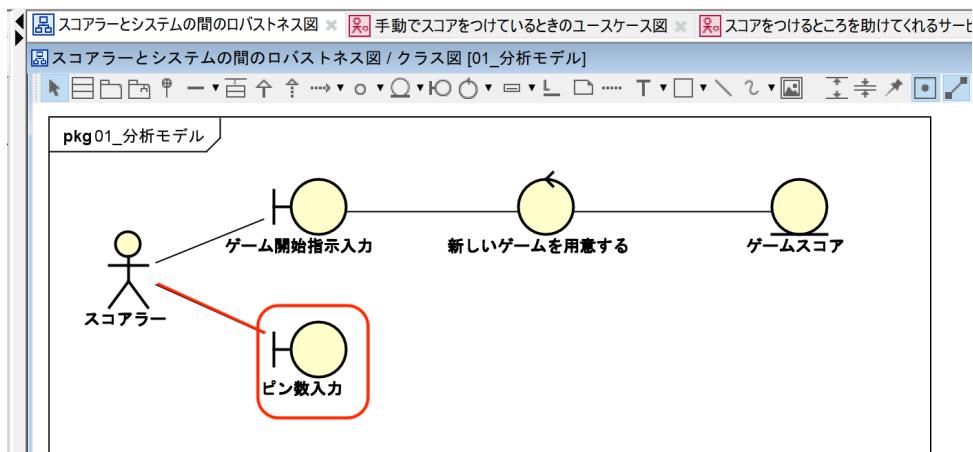


図 2.32 バウンダリ「ピン数入力」を追加する

#### 2. コントロール「ゲームを進める」を追加する( 図 2.33 )。

- パレットからコントロールのシンボルを選択し、ロバストネス図に追加する。
- 追加したコントロールの名前を「ゲームを進める」とする。
- バウンダリ「ピン数入力」とコントロール「ゲームを進める」の間に関連を引く。
- コントロール「ゲームを進める」とエンティティ「ゲームスコア」との間に関連を引く。

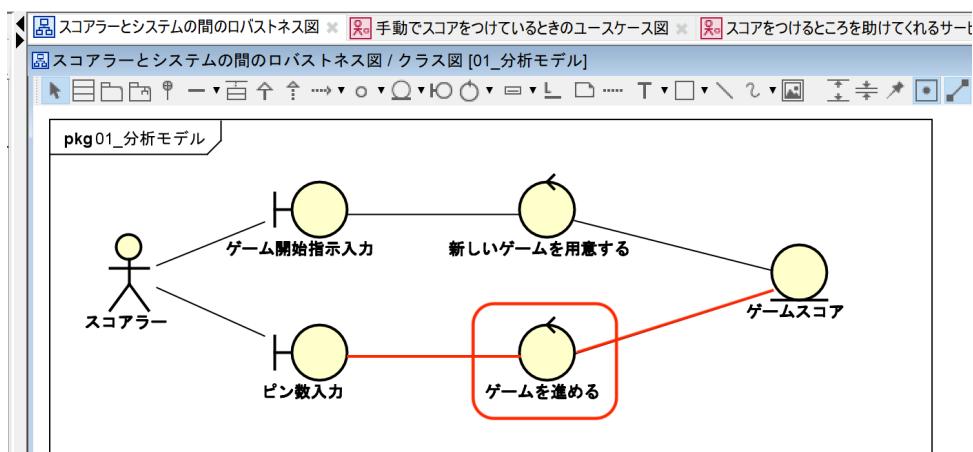


図 2.33 コントロール「ゲームを進める」を追加する

#### 3. バウンダリ「ゲームスコア出力」を追加する( 図 2.34 )。

- ・パレットからバウンダリのシンボルを選択し、ロバストネス図に追加する。
- ・追加したバウンダリの名前を「ゲームスコア出力」とする。
- ・アクター「スコアラー」とバウンダリ「ゲームスコア出力」の間に関連を引く。

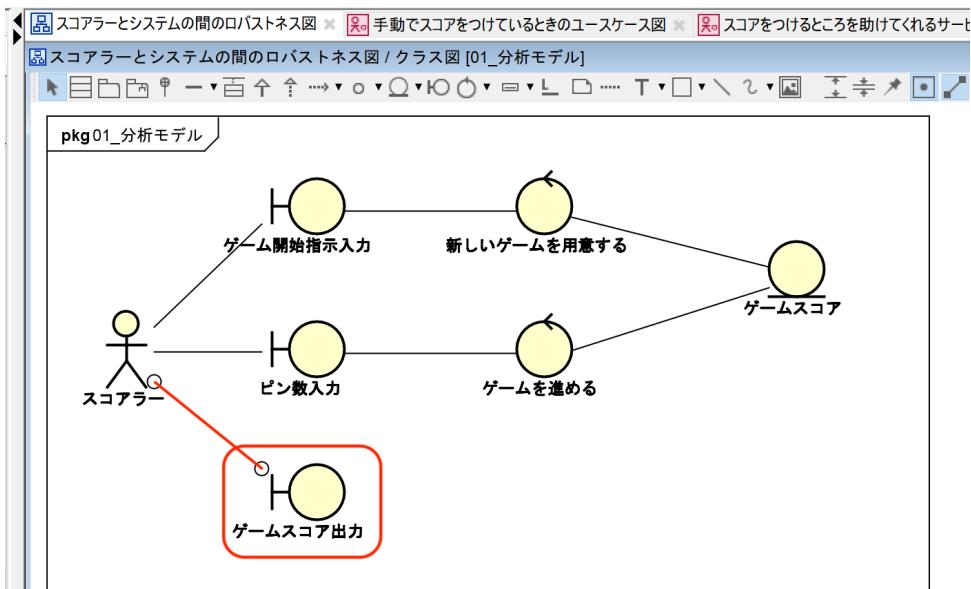


図 2.34 バウンダリ「ゲームスコア出力」を追加する

#### 4. コントロール「ゲームスコアを出力する」を追加する( 図 2.35 )。

- ・パレットからコントロールのシンボルを選択し、ロバストネス図に追加する。
- ・追加したコントロールの名前を「ゲームスコアを出力する」とする。
- ・バウンダリ「ゲームスコア出力」とコントロール「ゲームスコアを出力する」の間に関連を引く。
- ・コントロール「ゲームスコアを出力する」とエンティティ「ゲームスコア」の間に関連を引く。
- ・コントロール「ゲームを進める」とコントロール「ゲームスコアを出力する」の間に関連を引く。

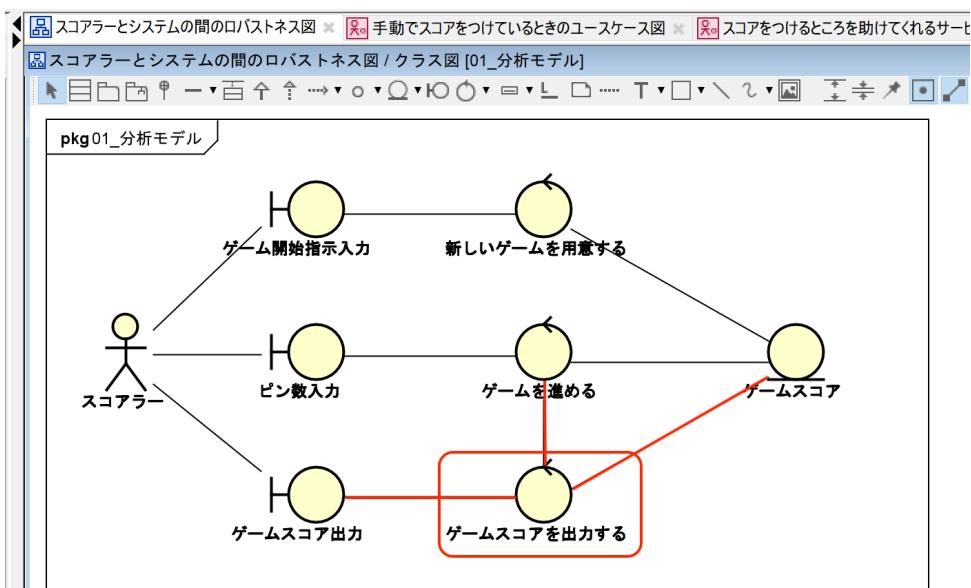


図 2.35 コントロール「ゲームスコアを出力する」を追加する

これで、スコアラーがピン数を入力したとき、ゲームを進め、ゲームスコアを更新するに必要な要素が洗い出せました。また、ゲームが進んだ際に、更新したゲームスコアを出力するのに必要な要素も洗い出せました。

## 2.3.4 スコアを計算する

スコアを記録したり、更新したスコアを出力できるようになりますが、まだスコアの計算を自動化するために必要な要素が洗い出せていません。そこで、こんどはスコアを計算する処理について検討してみましょう。

### スコアの計算の検討に使うロバストネス図を追加する

- モデルにクラス図を追加し、追加した図を「スコアの計算に関するロバストネス図」とする( 図 2.36 )。

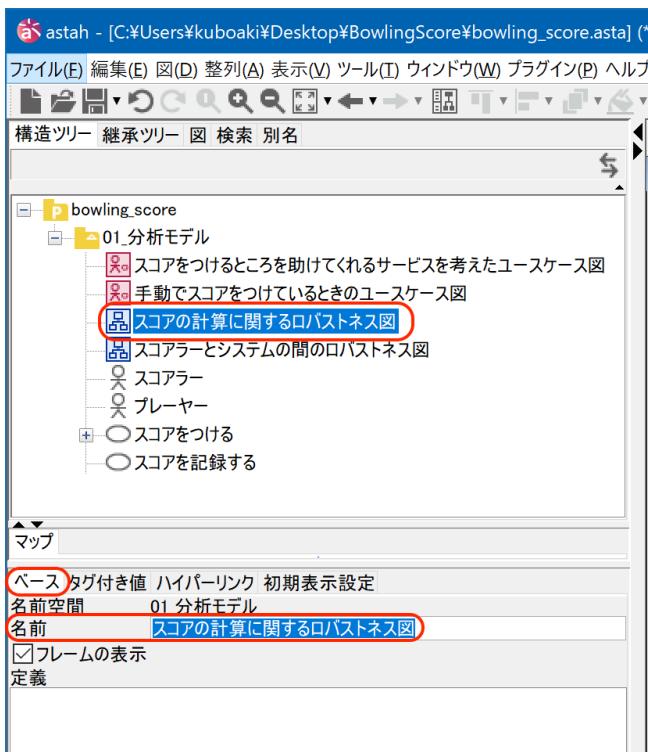


図 2.36 モデルにスコア計算に関するロバストネス図を追加する

スコアの計算に必要となりそうなエンティティとしては、「ゲームスコア」「現在のフレーム」「ゲームの状態」などがありそうです。ゲームの状態としては、1投目、2投目、サービスフレーム中、ゲームの終了といった状態が考えられます。

スコアの計算をするコントロールと、これらのエンティティを追加します。

### スコアを計算するコントロールと関連するエンティティを追加する

- コントロール「スコアを計算する」ならびに関連するエンティティを追加する( 図 2.37 )。
  - パレットからコントロールのシンボルを選択して追加し、「スコアを計算する」とする。
  - エンティティ「ゲームスコア」を構造ツリーからドラッグ&ドロップして追加する。
  - パレットからエンティティのシンボルを選択して追加し、「現在のフレーム」とする。
  - パレットからエンティティのシンボルを選択して追加し、「ゲームの状態」とする。
  - コントロール「スコアを計算する」から、それぞれのエンティティへ関連を引く。

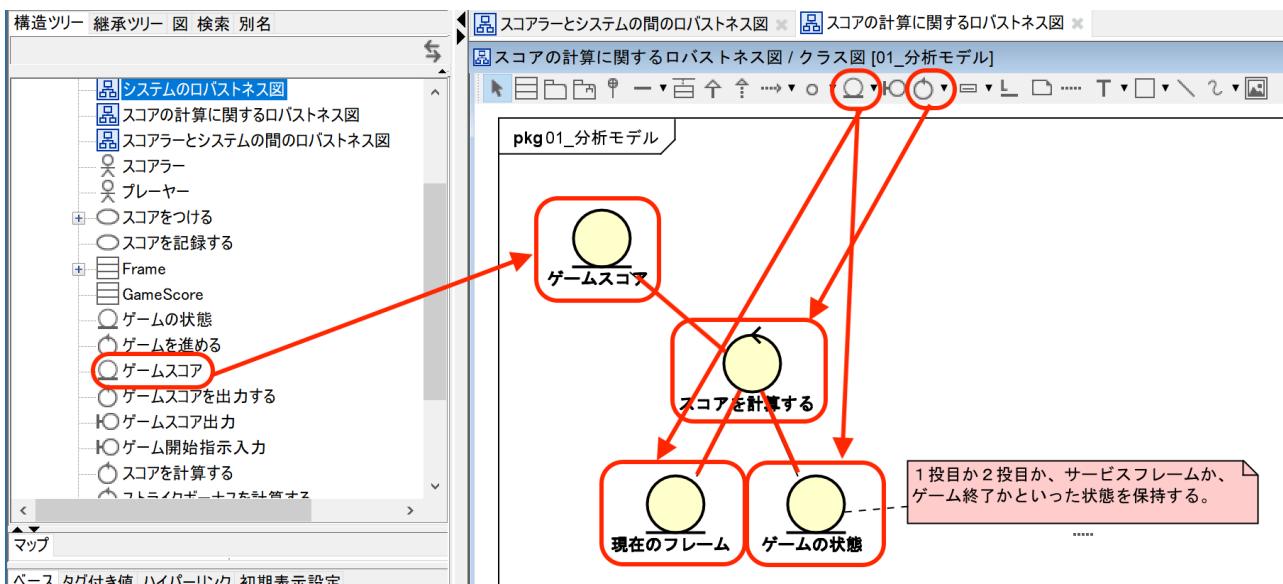


図 2.37 コントロール「スコアを計算する」を追加する

さらに、スコアを計算して記録するには、エンティティ「ゲームスコア」に対して計算の結果を記録するためのフレームを追加する処理が必要です。この処理をコントロールとして追加しましょう。スコアの計算では、ストライクボーナスとスペアボーナスの計算がありますが、これらも明示的に用意しておきましょう。

#### スコアを計算するコントロールと関連するコントロールを追加する

1. コントロール「フレームを追加する」を追加する( 図 2.38 )。
  - ・パレットからコントロールのシンボルを選択して追加し、「フレームを追加する」とする。
  - ・コントロール「スコアを計算する」からコントロール「フレームを追加する」へ関連を引く。
  - ・コントロール「フレームを追加する」からエンティティ「ゲームスコア」へ関連を引く。

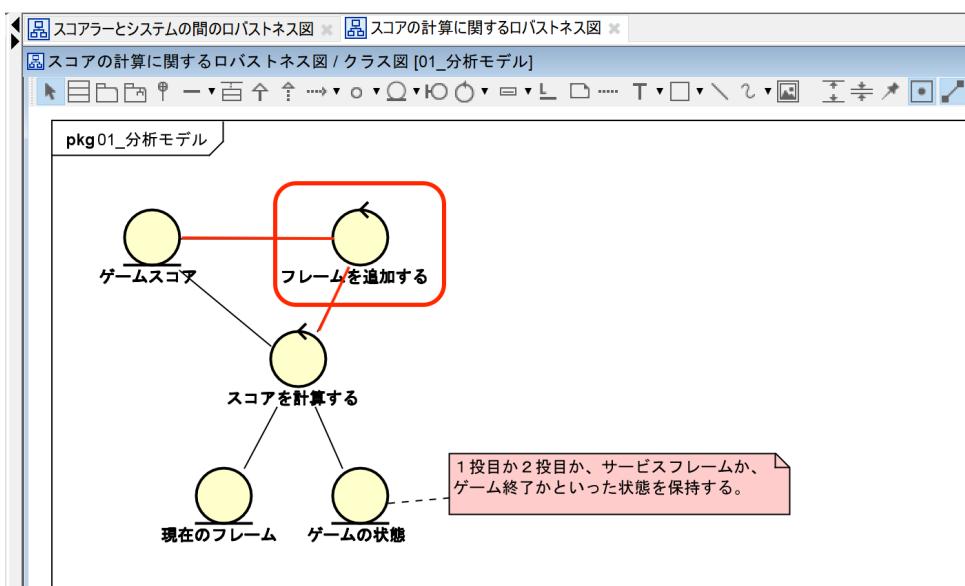


図 2.38 コントロール「フレームを追加する」を追加する

2. ボーナスの計算を追加する( 図 2.39 )。
  - ・パレットからコントロールのシンボルを選択して追加し、「ストライクボーナスを計算する」とする。

- ・パレットからコントロールのシンボルを選択して追加し、「スペアボーナスを計算する」とする。
- ・コントロール「スコアを計算する」からコントロール「ストライクボーナスを計算する」へ関連を引く。
- ・コントロール「スコアを計算する」からコントロール「スペアボーナスを計算する」へ関連を引く。
- ・コントロール「ストライクボーナスを計算する」からエンティティ「ゲームスコア」へ関連を引く。
- ・コントロール「スペアボーナスを計算する」からエンティティ「ゲームスコア」へ関連を引く。

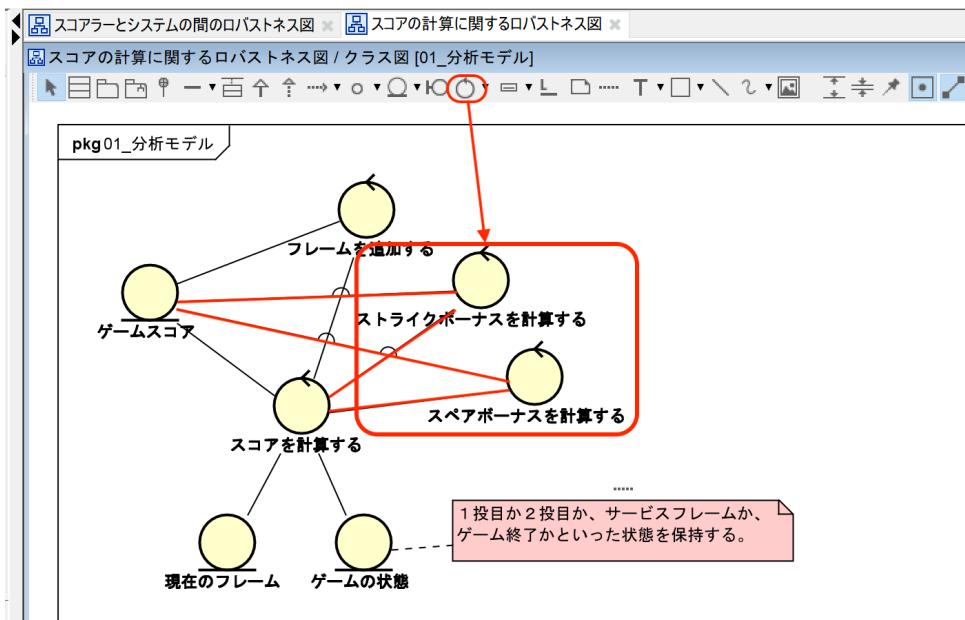


図 2.39 コントロール「ストライクボーナスを計算する」と「スペアボーナスを計算する」を追加する

これで、スコアを計算するのに必要な要素が洗い出せました。

## 2.3.5 システム全体を整理する

これまでに作成した「スコアラーとシステムの間のロバストネス図( 図 2.26 )」と「スコアの計算に関するロバストネス図( 図 2.36 )」をまとめると、システムのロバストネス図が作成できます。

エンティティ「ゲームスコア」は同じものですので、ひとつにまとめましょう。コントロール「スコアを計算する」はコントロール「ゲームを進める」の処理のなかで利用すると考えられますので、これらを関連づけます。

### 作成したロバストネス図をひとつにまとめる

1. モデルにクラス図を追加し、追加した図を「システムのロバストネス図」とする( 図 2.40 )。

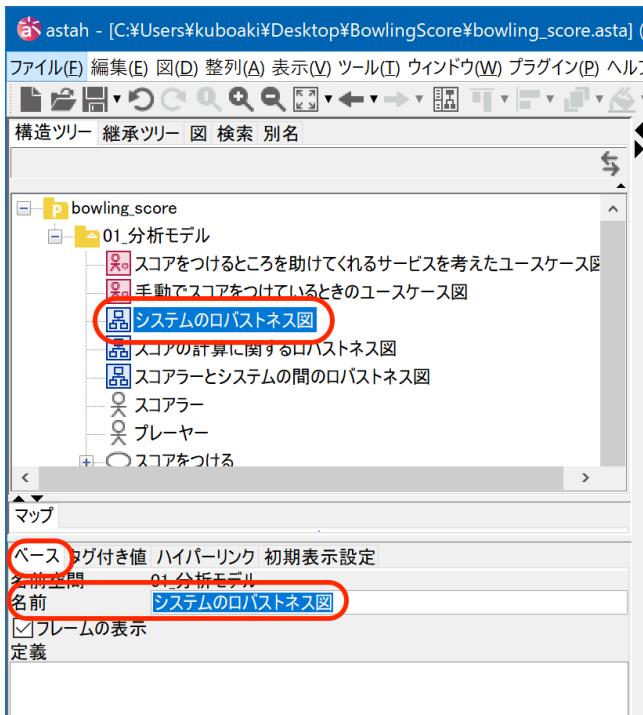


図 2.40 モデルにシステムのロバストネス図を追加する

2. ロバストネス図「スコアラーとシステムの間のロバストネス図」に記載した要素を取り込む。

- ロバストネス図「スコアラーとシステムの間のロバストネス図」を開き、作成した要素をすべてコピーする。
- ロバストネス図「システムのロバストネス図」を開き、ペーストする(図 2.41)。

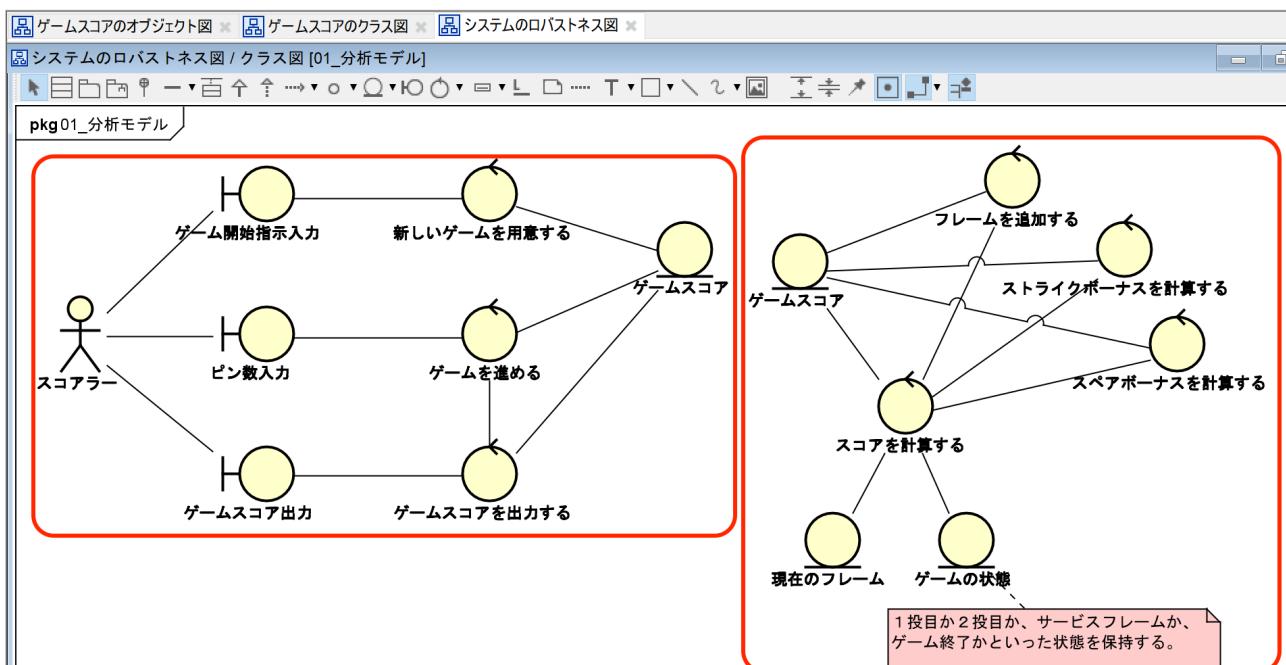


図 2.41 システムのロバストネス図に2つのロバストネス図をペーストする

3. ロバストネス図「スコア計算に関するロバストネス図」に記載した要素を取り込む。

- ロバストネス図「スコア計算に関するロバストネス図」を開き、作成した要素をすべてコピーする。

- ・ロバストネス図「システムのロバストネス図」を開き、ペーストする。
- ・2つあるエンティティ「ゲームスコア」のうち一方を選択し、右クリックしてポップアップメニューを開く。
- ・メニューから「図から削除」を選択し、エンティティ「ゲームスコア」のうち一方を消す( 図 2.42 )。

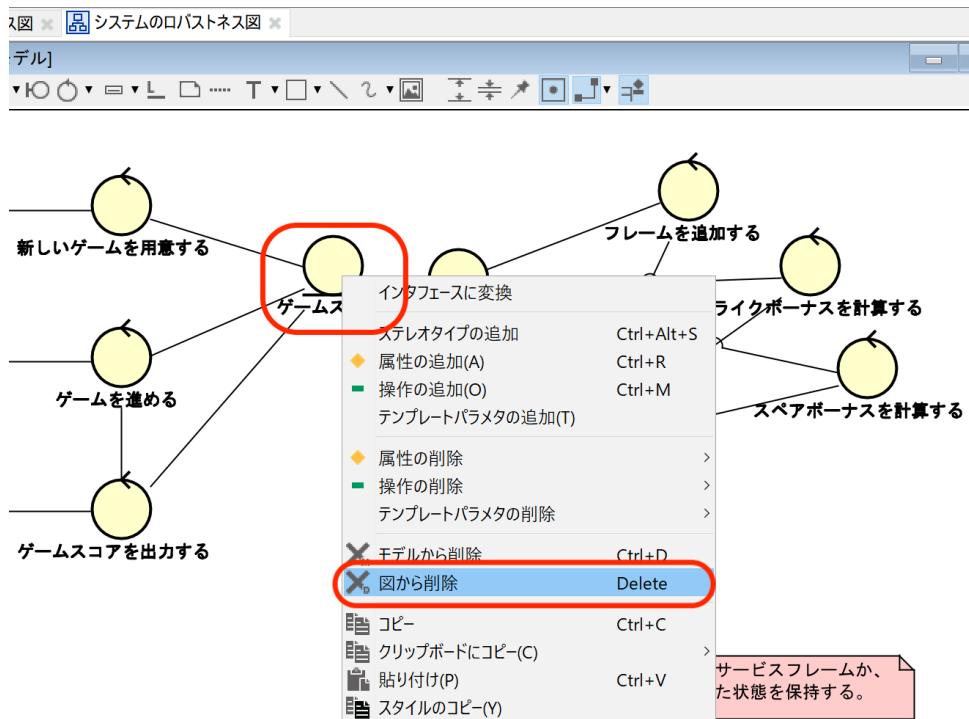


図 2.42 重複するエンティティ「ゲームスコア」の一方を消す

4. 残した方の「ゲームスコア」と他方につかがっていたコントロールの間に関連を引く( 図 2.43 )。

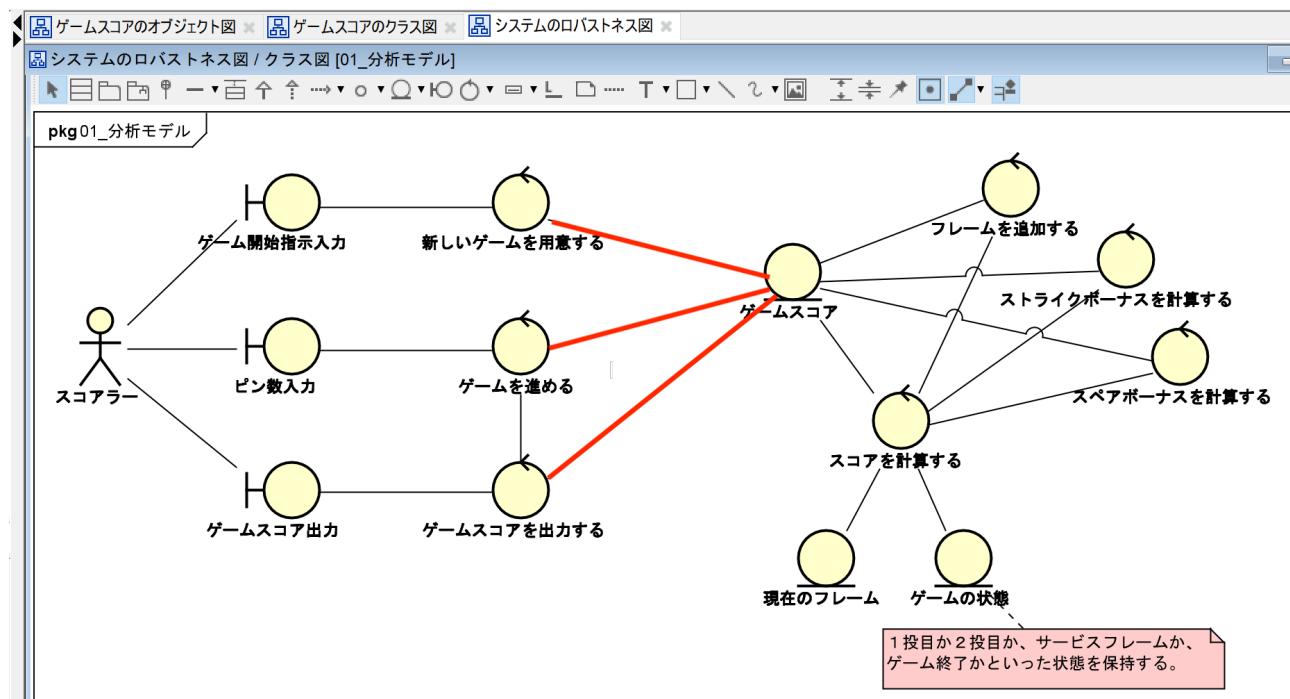


図 2.43 残したエンティティ「ゲームスコア」とコントロールの間に関連を引く

5. コントロール「ゲームを進める」とコントロール「スコアを計算する」の間に関連を引く。

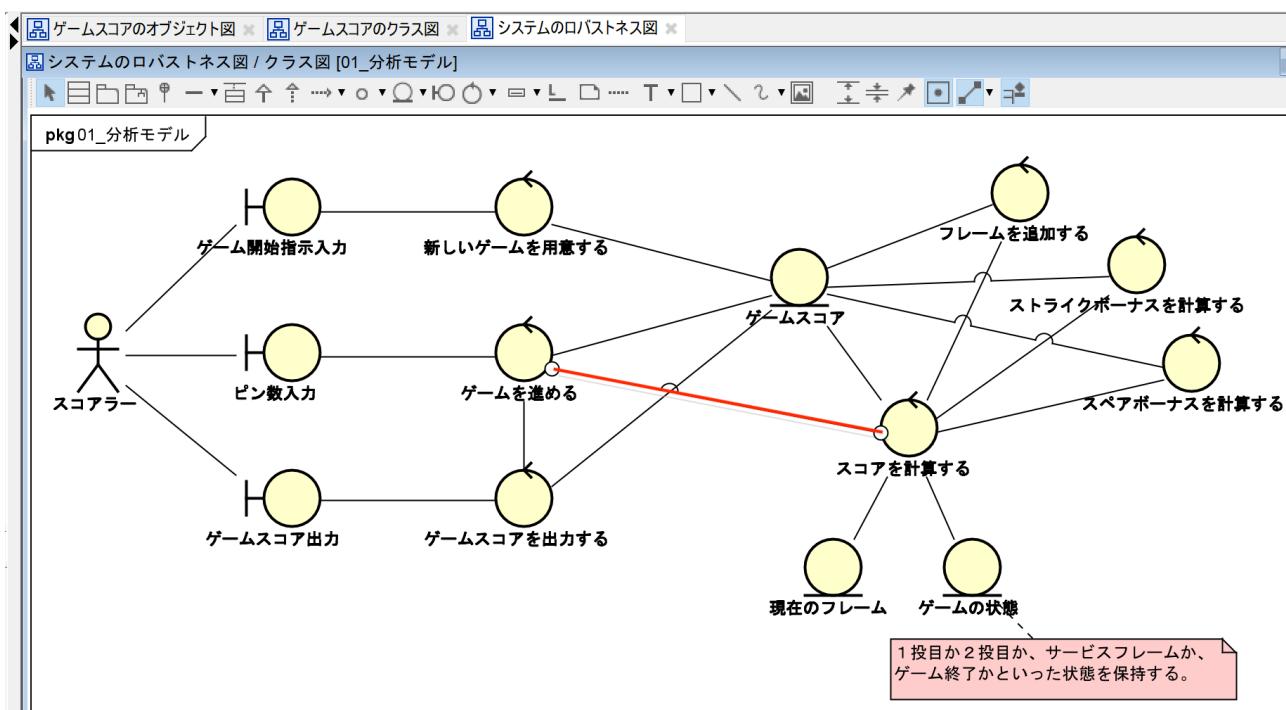


図 2.44 コントロール「ゲームを進める」とコントロール「スコアを計算する」ントロールの間に関連を引く

最後にできあがったロバストネス図から、このシステムにはどのような要素が必要なのかわかつてきました。

### 2.3.6 スコア計算を含むユースケース記述を書く

作成したロバストネス図に登場する要素を参考にして、ユースケース記述を見直してみましょう。

ロバストネス図を作成した結果、ユースケース「スコアを記録する」には、次のような働きが含まれていることがわかりました。

- ・スコアを計算する
- ・(ゲームスコアに)フレームを追加する
- ・ストライクボーナスを計算する
- ・スペアボーナスを計算する

このことを加味して、ユースケース「スコアを記録する」のユースケース記述(リスト2)を見直してみましょう(リスト3)。

### リスト 3. ユースケース「スコアを記録する」のユースケース記述(見直し後)

1. スコアラーがゲームの開始指示を出すと、システムは新しいゲームを用意する。
2. スコアラーは、プレーヤーが投球した結果をみてピン数をシステムに入力する。
3. システムは、ピン数を受け取ると、ゲームを進め、ユースケース「スコアを計算する」を実行する。
  - a. 現在の状態が、1投目のピン数入力待ちのとき:
    - ゲームスコアに新しいフレームを追加する。
    - 現在のフレームの1投目にピン数を記録する。
    - 1投目がストライクだったときは、現在の状態を次の1投目のピン数入力待ちに変更する。
    - ストライクでなかったときは、現在の状態を2投目のピン数入力待ちに変更する。
  - b. 現在の状態が、2投目のピン数入力待ちのとき:
    - 現在のフレームの2投目にピン数を記録する。
    - 現在の状態を次の1投目のピン数入力待ちに変更する。
4. スペアボーナスを計算し、ゲームスコアに反映する。
  - a. 前のフレームがスペアのとき:
    - 1投目のピン数を前のフレームのスペアボーナスに記録する。
5. ストライクボーナスを計算し、ゲームスコアに反映する。
  - a. 前のフレームがストライクのとき:
    - 1投目と2投目のピン数の合計を前のフレームのストライクボーナスに記録する。
  - b. 前の前のフレームがストライクのとき:
    - 1投目のピン数と前のフレームの1投目のピン数を、前の前のフレームのストライクボーナスに記録する。
6. 10フレームになったときは、サービスフレームを追加する。
7. 上記2から6を、10フレームになるまで繰り返す。

## 2.3.7 10フレーム目の扱いについて検討する

「クラッシックスコアリング」の場合、10フレーム目が他のフレームとは異なっています。サービスフレームという考え方があり、ストライクやスペアの場合に追加で投球できます。

このサービスフレームの扱い方について整理しておきましょう。

### 10フレーム目がストライクもスペアもない場合

10フレーム目がストライクでもスペアもない場合、9フレーム目までの通常のフレームと同様、1投目と2投目を記録するだけです( 図 2.45 )。サービスフレームは追加されません。

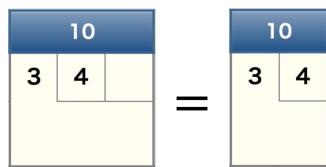


図 2.45 10フレーム目がストライクもスペアもない場合

## 10フレーム目がスペアの場合

10フレーム目の2投目でスペアになった場合、1投目と2投目を通常のフレームと同様に記録したのち、もう1投追加されます。これを、10フレーム目に加えて、11フレーム目の1投目が追加されたとみなします。つまり、ゲームスコアのデータを「図 2.46」のように構成します。

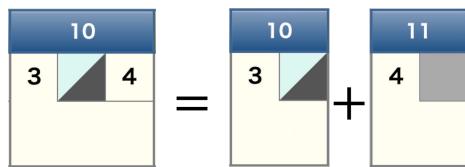


図 2.46 10フレーム目がスペアの場合

ゲームスコアをこのように構成しておくと、10フレーム目も、通常フレームの組み合わせによってボーナスも計算できます。ゲーム終了時の10フレームのスコア(ピン数に以後の投球によるボーナスを加えたスコア)を取得するとそれがゲームのスコアになります。

## 10フレーム目の1投目がストライクの場合

10フレーム目の1投目がストライクの場合、10フレーム目をストライクとした上で、もう2投追加されます。これを、10フレーム目に加えて、11フレーム目の1投目と2投目が追加されたとみなします。つまり、ゲームスコアのデータを「図 2.47」のように構成します。ただし、11フレーム目の1投目がストライクの場合は、同じ2投追加でも次の「2投目もストライクの場合」の考え方を使います。

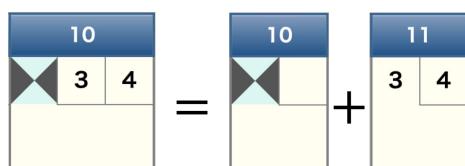


図 2.47 10フレーム目の1投目がストライクの場合

ゲームスコアをこのように構成しておくと、10フレーム目も、通常フレームの組み合わせによってボーナスも計算できます。ゲーム終了時の10フレームのスコア(ピン数に以後の投球によるボーナスを加えたスコア)を取得するとそれがゲームのスコアになります。

## 10フレーム目の2投目もストライクの場合

10フレーム目の2投目もストライクの場合、10フレーム目、11フレーム目をストライクとした上で、もう1投追加されます。これを、10フレーム目、11フレーム目に加えて、12フレーム目の1投目が追加されたとみなします。つまり、ゲームスコアのデータを「図 2.48」のように構成します。このように構成すれば、通常のフレームでダブルをとったときの計算方法(ストライクが続いたときはさらに次のフレームの1投目が2投目として加算される)のまで10フレーム目のボーナスが計算できます。

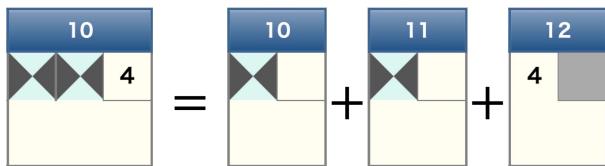


図 2.48 10フレーム目の2投目もストライクの場合

ゲームスコアをこのように構成しておくと、10フレーム目も、通常フレームの組み合わせによってボーナスも計算できます。ゲーム終了時の10フレームのスコア(ピン数に以後の投球によるボーナスを加えたスコア)を取得するとそれがゲームのスコアになります。

## 2.4 解決したいのはゲームの進行とスコアの計算

ユースケース吟味した結果、自動化によって解決したかったのは、ゲームの状態を把握することで「ゲームを進める」ときのスコアラーの入力を簡便にすることだとわかりました。また、「スコアを計算する」とこと、とりわけ「ストライクボーナスやスペアボーナスのを計算すること」だとわかりました。

このような検討結果から、このチュートリアルでは「スコアを記録する」というユースケースを実現することを目標にします。とくに、「ゲームを進める」とことと「ボウリングのスコアを計算する」ことについて、モデルを使って表現し、そのモデルからプログラムを得るように努めます。

スコアのつけ方については、これまで検討してきた「クラシックスコアリング」を対象としましょう。「カレントフレームスコアリング」に比べて、フレームの間に依存関係がある分、スコア計算がより複雑になっています。



# 3 スコアの構造を図にする

ここからは、スコアを計算するのに使う要素と、要素同士の関連、ゲームを進める動作をモデル図を使って整理してみましょう。

## 3.1 ゲームスコアのオブジェクト図を描く

吟味したユースケース記述を参考しながら、実際のゲームの様子を具体的な要素を使って図に表してみることで、スコアの計算にはどんな要素があればよいか考えてみましょう。

具体的な要素とは、ボウリングでいえば、ゲームやその各フレーム、ピンの数、スコアやボーナスを指します。このようなことを図で表すには「オブジェクト図（インスタンス図と呼ぶこともあります）」が向いています。

### 3.1.1 プロジェクトに設計モデルを追加する

まず、プロジェクトに設計モデルを追加しましょう。ここで言う設計モデルは、設計用に作成するモデル図を格納する場所だと思ってください。



ここでいう設計とは、対象とする業務やサービスに必要となる構成要素を洗い出すことと、それらを使った振舞いを検討することだと考えておけばよいでしょう。

#### プロジェクトに設計モデルを追加する

1. プロジェクトにモデルを追加する。
  - ・構造ツリー上で、プロジェクトを選択する。
  - ・右クリックしてポップアップメニューを開き、「モデルの追加>モデル」を選択する（図 3.1）。

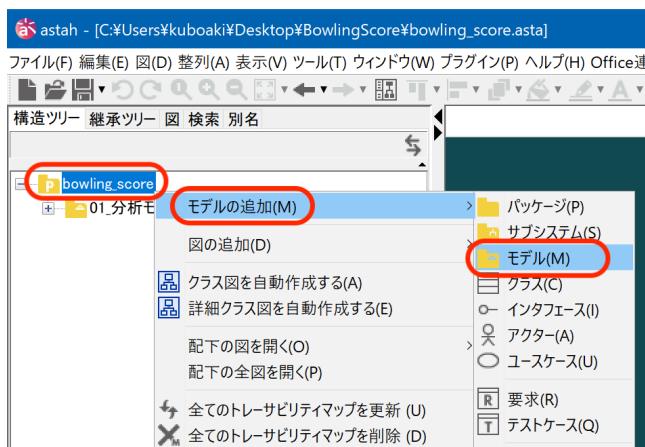


図 3.1 プロジェクトにモデルを追加した

## 2. 追加したモデルに名前をつける

- 構造ツリー上で、追加したモデルを選択した状態で、プロパティの「ベース」タブを選択する。
- 「名前」を編集して「02\_設計モデル」とする( 図 3.2 )。

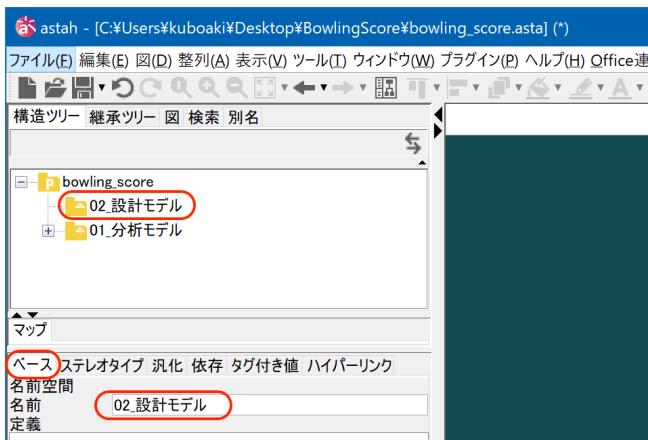


図 3.2 モデルに設計用のモデルとして名前をつけた

### 3.1.2 設計モデルにオブジェクト図を追加する

ゲームスコアのオブジェクトが作られていく様子をオブジェクト図で表してみましょう。astah\* で「オブジェクト図」を作成するときは「クラス図」を使います。

#### ゲームスコアのオブジェクト図を追加する

- モデルにクラス図を追加する( 図 3.3 )。

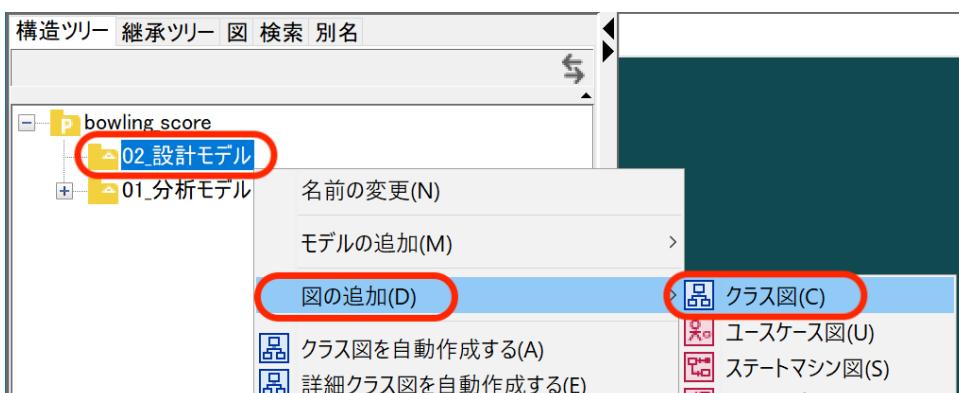


図 3.3 モデルにクラス図を追加する

- 追加した図を「ゲームスコアのオブジェクト図」とする( 図 3.4 )。

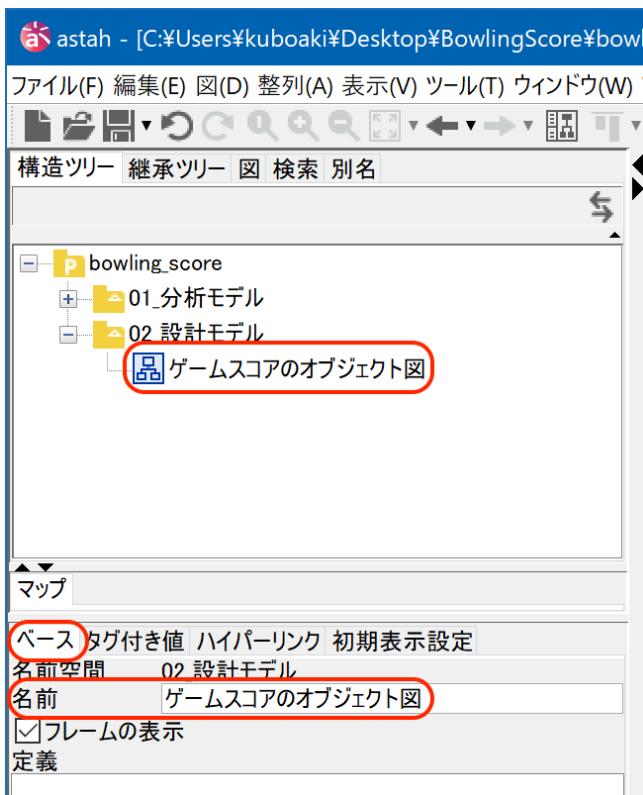


図 3.4 「ゲームスコアのオブジェクト図」を追加する

### 3.1.3 ゲームのオブジェクトを作成する

ユースケース記述を読むと、「ゲーム」と「フレーム」のオブジェクトが必要そうです。まず、ゲームを表すオブジェクトを追加しましょう。

#### ゲームを表すオブジェクトを追加する

1. パレットから「インスタンス仕様」を選択して、図に配置する( 図 3.5 )。

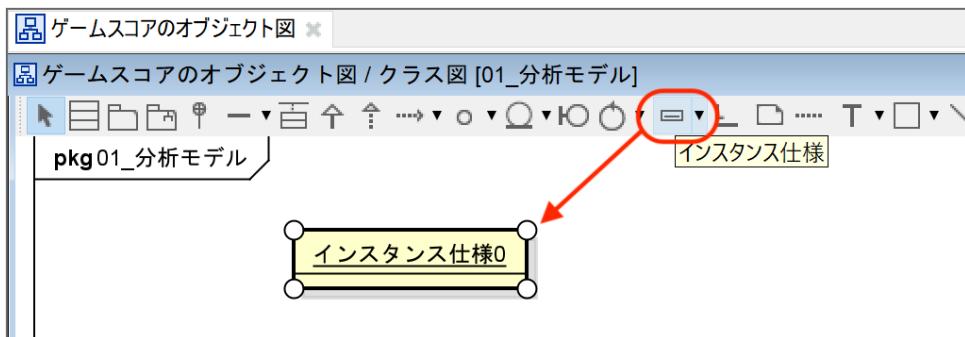


図 3.5 「インスタンス仕様」を追加する

2. 名前が「インスタンス仕様0」(数字は作るたびにつけられる)となっている。これを、具体的なゲームを表す名前に変える。ここでは「game1」とする( 図 3.6 )。

- ・これはこのオブジェクト図で表そうとしているゲームにつけた識別子。
- ・名前の文字列を直接クリックして編集してもよいし、オブジェクトを選択した状態でプロパティから編集してもよい。

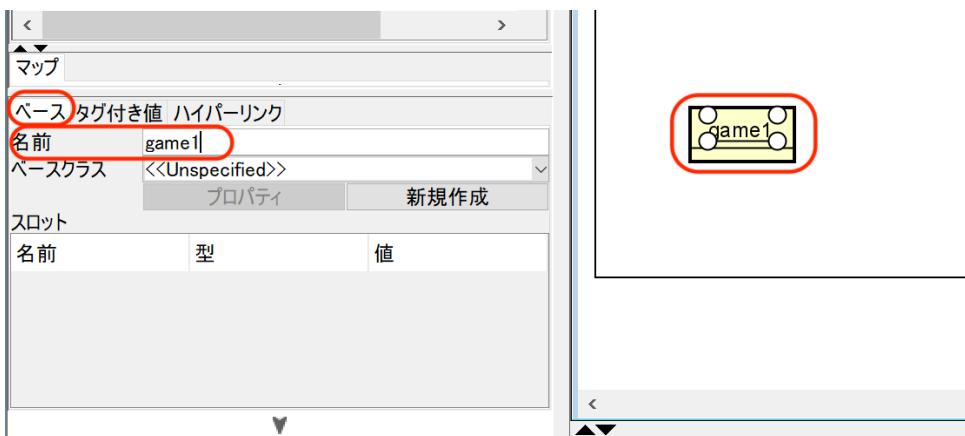


図 3.6 名前を「game1」に変更する

3. このオブジェクトを、ゲームを表す「Game」クラスのオブジェクトとするために「Game」を追加する。
- 「game1」を選択した状態で、プロパティから「新規作成」ボタンをクリックする( 図 3.7 )。
  - クラスを定義するダイアログが表示されるので、クラスの名前を「GameScore」にする( 図 3.8 )。

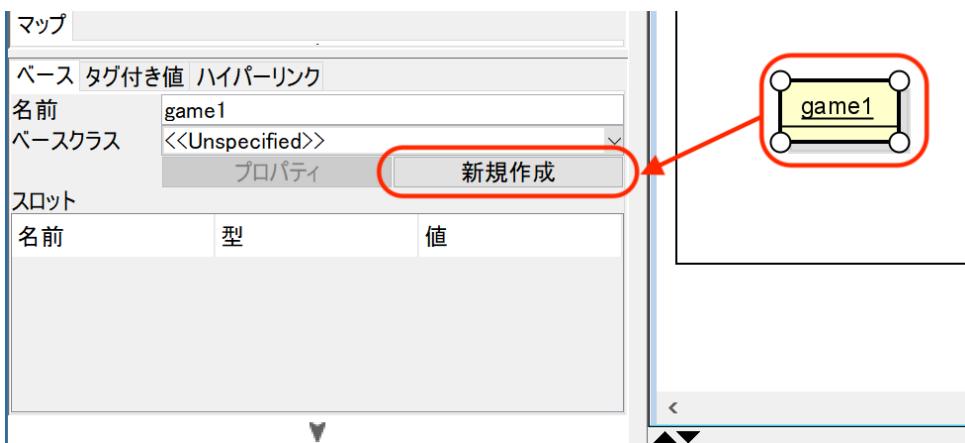


図 3.7 オブジェクトを選択してクラスを追加する

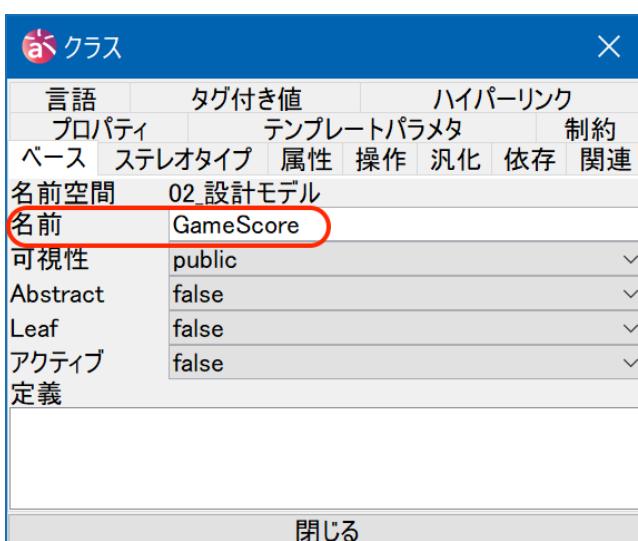


図 3.8 クラス「GameScore」を定義する

4. オブジェクトの表示も「game1 : GameScore」のように変わる( 図 3.9 )。

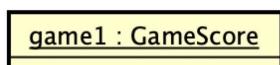


図 3.9 クラスを追加した結果「game1」が「game1:GameScore」に変わった

### 3.1.4 フレームのオブジェクトを作成する

ゲームと同じように、フレームについてもインスタンス仕様を追加して、クラス名として「Frame」をつけます。また、ユースケース記述から、1投目と2投目のピン数、スペアボーナスとストライクボーナスを保持する必要がありそうだとわかっています。これらは、個別のインスタンス仕様と考えることもできますが、ここでは Frame に従属する情報と考えて属性にしてみます。

フレームを表すオブジェクトを追加し、「Frame」クラスと属性を定義する

1. パレットから「インスタンス仕様」を図に追加する。
2. フレーム名として「01」、クラス名として「Frame」をつける( 図 3.10 )。

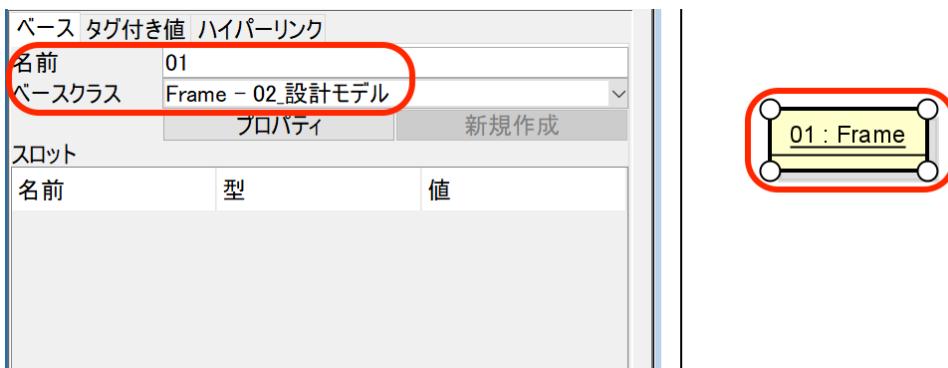


図 3.10 フレームオブジェクトと「Frame」クラスを追加する

2. フレームオブジェクトを選択した状態で、プロパティボタンをクリックする( 図 3.11 )。

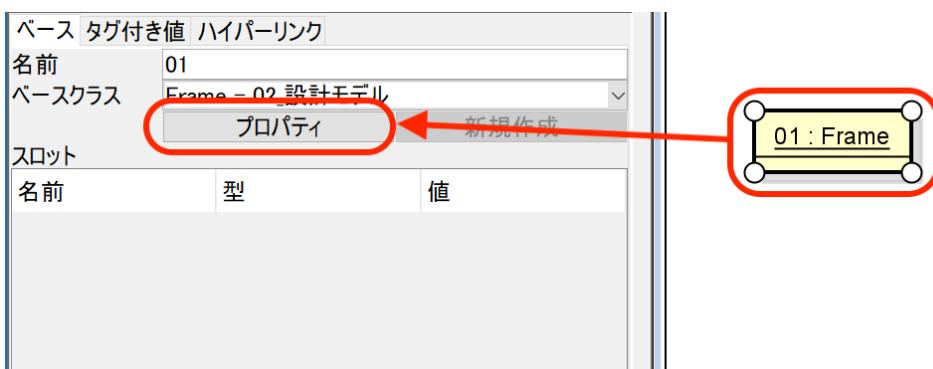


図 3.11 「Frame」クラスのプロパティダイアログを開く

3. 「Frame」クラスに従属する情報をクラスの属性に追加する( 図 3.12 )。

- 「属性」タブを開く。
- 「+」ボタンで属性を追加し、名前を入力する。
- 「first」「second」「spare\_bonus」「strike\_bonus」を定義する。

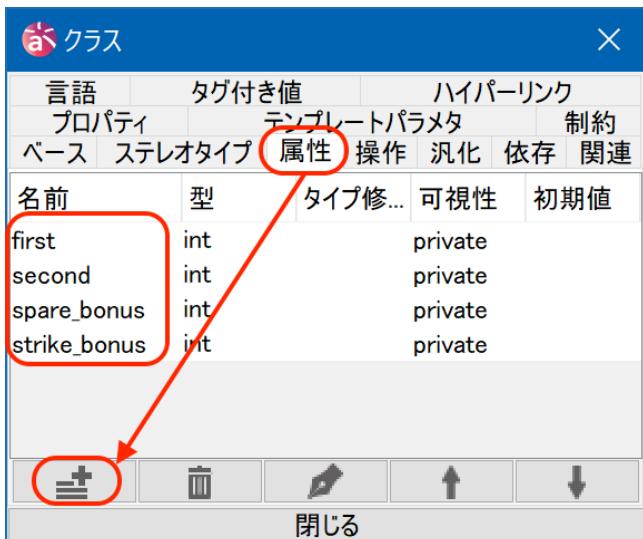


図 3.12 「Frame」クラスに従属する情報を属性に追加した

4. 追加したオブジェクトは 図 3.13 のように変わる。

- 属性は追加されたが、このオブジェクトには属性値はまだ設定されていない。

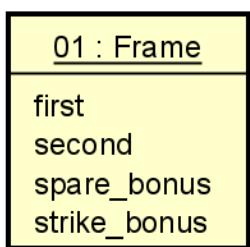


図 3.13 属性が追加された後の「Frame」クラスのオブジェクト

5. 作成したフレームオブジェクトを選択すると、属性のインスタンス(スロットと呼ぶ)を編集するプロパティが表示される( 図 3.14 )。

- 各属性の「値」の欄をダブルクリックすると、値が設定できるようになる。

6. オブジェクトに属性値を設定する。

- 1ストライクなので投目(first)を 10 にする。
- 2投目(second)は投球していないが、ここでは 0 としておく。
- ボーナスは未確定だが、これも 0 としておく。

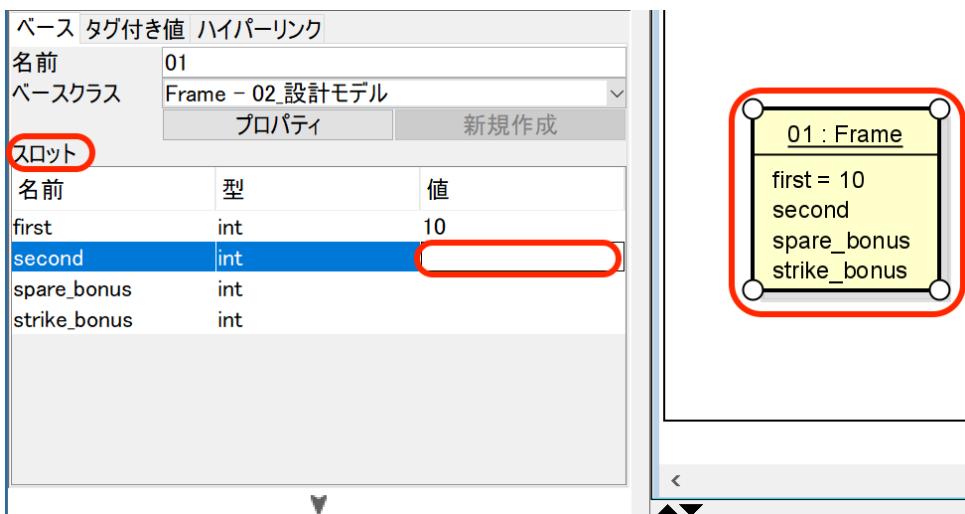


図 3.14 オブジェクトの属性値のプロパティを開く

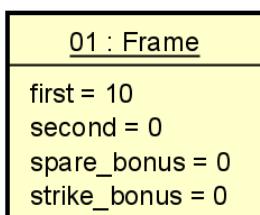


図 3.15 オブジェクトに属性値が設定された

もちろん、別の捉え方をしているのであれば、オブジェクトや保持する属性などが異なってきます。この演習では、図 3.15 のように捉えて、以降の設計を進めます。

### 3.1.5 ゲームの進行に合わせてオブジェクトを追加する

続いて、進行中のゲームのスコア(図 3.16)を例として、オブジェクト図にインスタンスを追加してみます。

名前	1	2	3	4	5	6	7	8	9	10	Total
くぼあき	20	38	47	75	94	103					

図 3.16 第6フレーム投球後のスコアの例

各フレームの情報を格納するデータ領域は、1ゲーム分を先に用意しておく方法や、ゲームの進行に応じて1フレームずつ増やす方法もあります。ここでは、1フレームずつ追加する方法を使ってみましょう。また、前後のフレームとのつながりを保持する方法にも、配列やリストを使う方法があります。ここでは、連結リスト(Linked List)を使ってみることにしましょう。

そして、ゲームからみて「先頭のフレーム」と「現在投球中のフレーム」がわかるよう、「game1」オブジェクトからは、それぞれ別のリンクを張って、辿れるようにしておきます。

ではまず、ここまでに作成した「GameScore」クラスのオブジェクト「game1」と「Frame」クラスの1フレーム目のオブジェクト「01」の間にリンクを引くところから始めましょう。

### オブジェクトの属性値を設定する

1. ゲームからフレームへリンク(関連のインスタンス)を引く。

- ・パレットからリンクを選択し「game1:GameScore」にカーソルを移動して青い枠を出す( 図 3.17 )。
- ・マウスをドラッグして「01:Frame」へリンクを伸ばし、青い枠が出たらマウスのボタンを離すとリンクが引ける( 図 3.18 )。

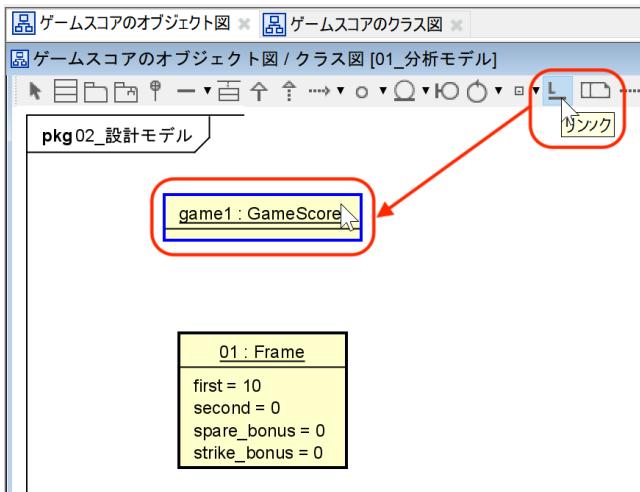


図 3.17 ゲームからフレームへリンクを引く

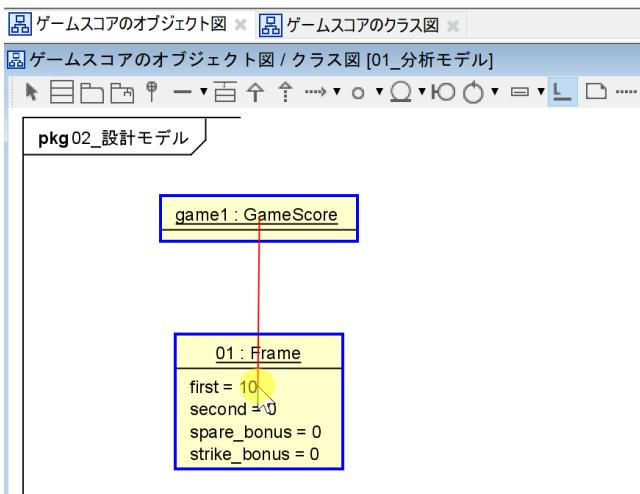


図 3.18 ゲームからフレームへリンクを引いた

2. ゲームはフレームを参照するので、誘導可能の矢印をつける。逆は未確定なので誘導可能性を未確定(矢印をつけない)にしておく。

- ・リンクのフレーム寄りでマウスの右クリックして、「誘導可能性>誘導可能」を選択する( 図 3.19 )。
- ・リンクに誘導可能を示す矢印が引かれる( 図 3.20 )。



図 3.19 ゲームからフレームへのリンクを誘導可能にする

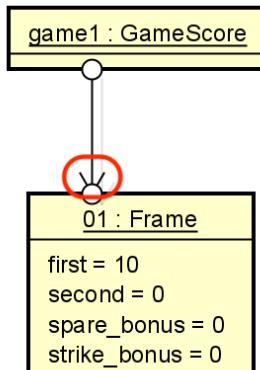


図 3.20 ゲームからフレームへのリンクを誘導可能にした

3. ゲームがフレームを参照するときに使うリンク端に名前をつける。ここでは先頭のフレームを指すリンクなので、リンク端名「head」をつける。

- リンクのフレーム寄りでマウスの右クリックして、「リンク端の設定」を選択する( 図 3.21 )。
- リンク端に、現在つながっているインスタンスの名前「01」割り当てられる。これを「head」に変更する( 図 3.22 )。

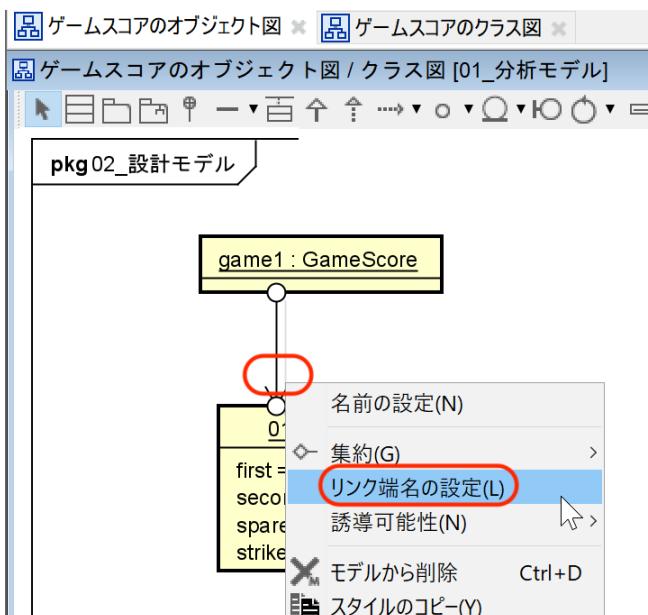


図 3.21 フレーム側のリンク端に名前をつける

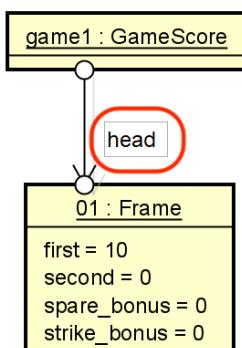


図 3.22 フレーム側のリンク端に名前をつけた

4. 同様にして、現在のフレームを指すリンクを追加し、リンク端名「current」をつける( 図 3.23 )。

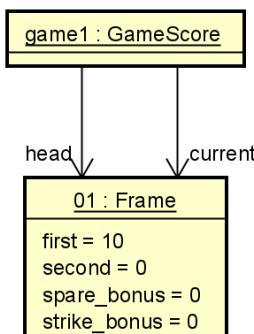


図 3.23 別のリンク「current」を追加した

第2フレームを追加しましょう。追加するフレームのピン数は、図 3.16 を参照してください。

第2フレームを追加し、ピン数を設定する

1. パレットから「インスタンス仕様」を図に追加する。
2. 追加したインスタンス仕様の名前を「02」を入力すると、図に反映される( 図 3.24 )。
3. ベースクラスのプルダウンメニューから「Frame」を選択する。

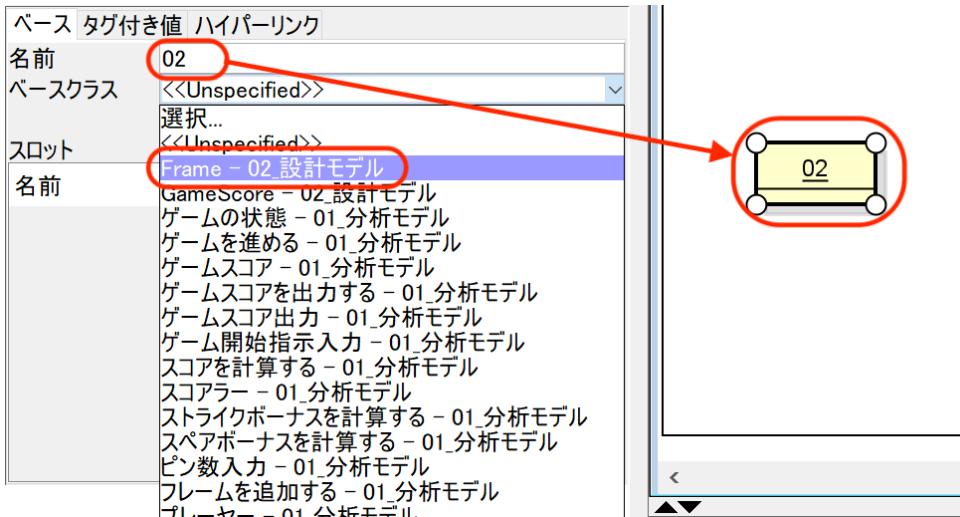


図 3.24 インスタンス仕様を追加して名前とクラスを設定した

3. Frameクラスに設定後は、プロパティにピン数などのスロットが表示される。
4. 図 3.16 を参照して、1投目、2投目のピン数を設定する( 図 3.25 )

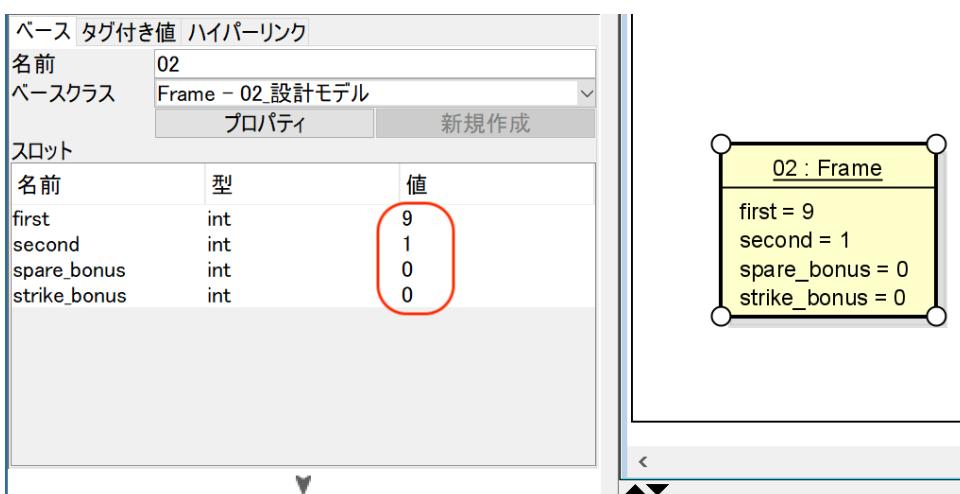


図 3.25 追加したフレームにピン数を設定した

前後のフレームとのつながりには連結リスト(Linked List)を使ってみることにしました。連結リストになるよう、追加した第2フレームと第1フレームの間にリンク(関連のインスタンス)を追加しましょう。

#### フレームの間にリンクを追加する

1. パレットからリンクを選択し、「01:Frame」から「02:Frame」へリンクを引き、誘導可能にする( 図 3.26 )。

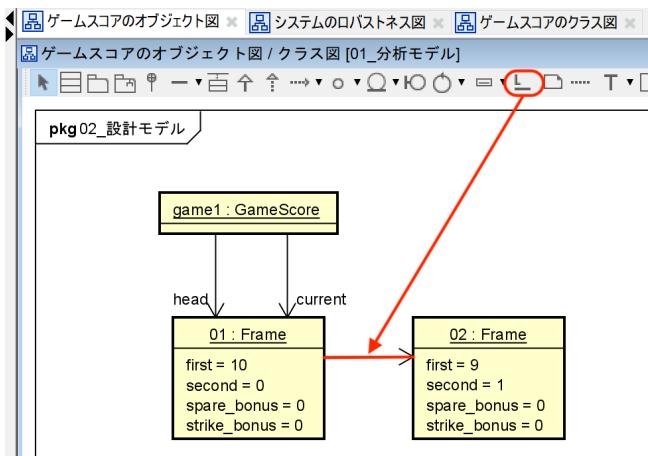


図 3.26 フレームの間にリンクを追加し、誘導可能にした

2. 追加したリンクを選択した状態でプロパティを編集する( 図 3.27 )。

- ターゲットが「02」になっているリンク端のタブを選択する。
- リンク端の名前を「next」とする。

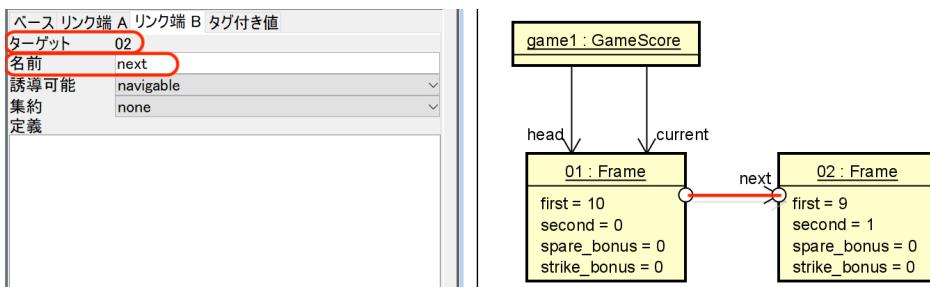


図 3.27 第2フレーム側のリンク端の名前を「next」とした

3. 同様にして、previousを意味するリンク「prev」も追加する( 図 3.28 )。

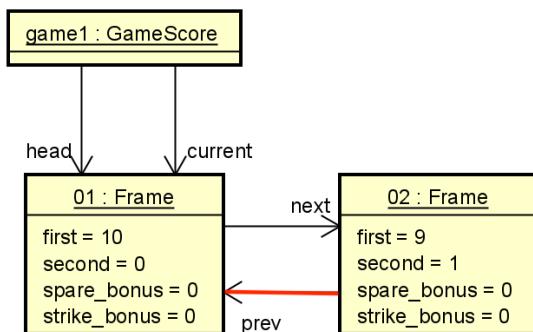


図 3.28 第1フレームへ向かうリンクを追加して、リンク端の名前を「prev」とした

4. 第1フレームの「prev」、第2フレームの「next」のつながる先はないので、それぞれについてインスタンス仕様から「nil」を追加しリンクを追加しておく( 図 3.29 )。

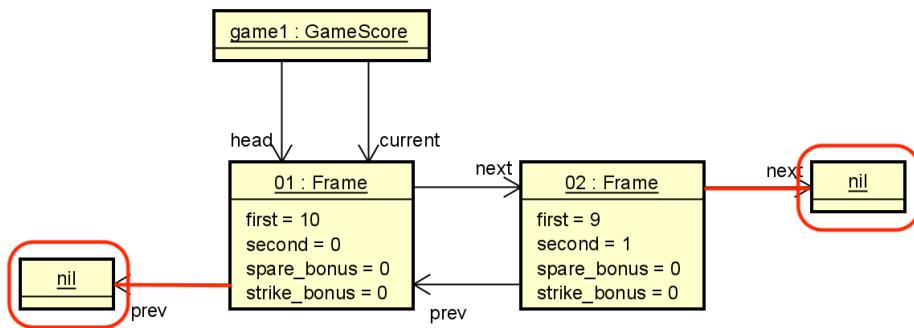


図 3.29両端に「nil」へのリンクを追加した

5. 現在投球中のフレームを指すリンク「current」は、第2フレームを指すように変更する( 図 3.30 )。

- リンクを選択し、誘導可能なリンク端(「current」と書いてある側)をマウスでドラッグする。
- マウスカーソルを第2フレームの枠内へ移動してドロップすると、接続先を変更できる。

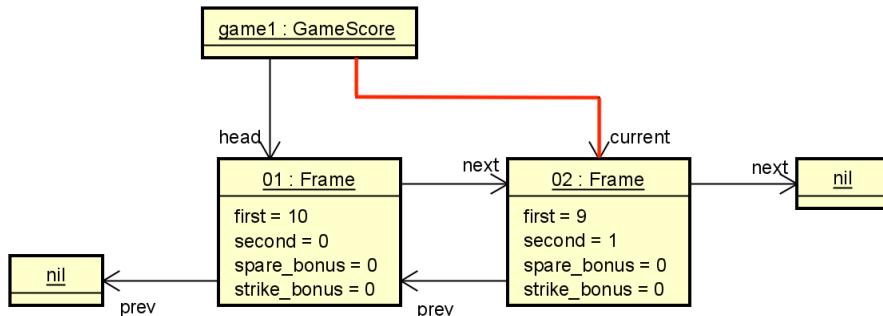


図 3.30「current」が第2フレームを指すように変更した

6. 第1フレームはストライクだったので、第2フレームの2投分によってストライクボーナスを計算してスロットの値に反映する( 図 3.31 )。

- スペアボーナスとストライクボーナスに設定する値は、ユースケース記述「リスト3」に記述した方法を使って求める。

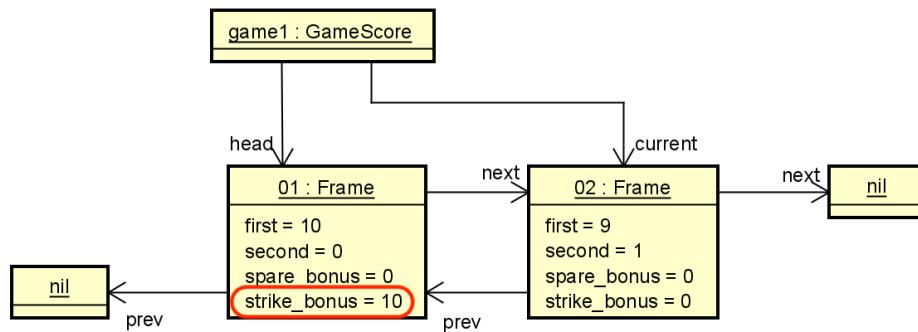


図 3.31 第1フレームのストライクボーナスを計算して、スロット値を更新した

これで、第2フレームまで投球した場合のオブジェクト図ができました。同じようにして、図 3.16 を参照しながら、第6フレーム投球後のオブジェクト図を作成してみましょう( 図 3.32 )。

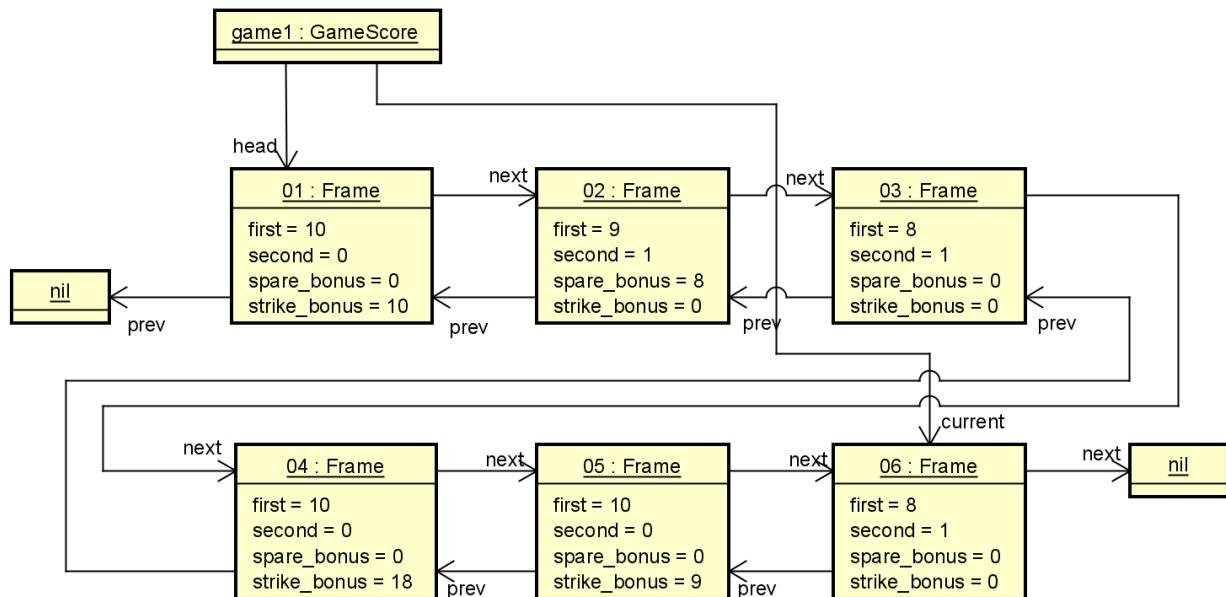


図 3.32 第6フレーム投球後のオブジェクト図

オブジェクト図を作成してみることで、スコアを記録するために必要なクラスや、クラスが持つ属性などを検討できましたね。

また、ストライクやスペアのボーナスには「次のフレーム」あるいは「次の次フレーム」のピン数が影響します。ユースケースに従ってオブジェクト図にオブジェクトを追加することで、現在のフレームの1投目あるいは2投目にピン数を入力されたときが、「前のフレーム」や「前の前のフレーム」のボーナスを計算するタイミングということもわかつてきました。



構成要素のどの部分をインスタンス仕様とし、どの部分をインスタンス仕様の属性として扱うかによって、作成されるオブジェクト図は変わってきます。図 3.32 は、何通りも作成しうるオブジェクト図の作成例の1つです。

より実務的な検討においては、複数の捉え方を考えた上でその考えに基くオブジェクト図を作成し、それらの中からより妥当と考えられるものを選ぶべきでしょう。

## 3.2 ゲームスコアのクラス図を描く

オブジェクト図が作成できたので、これを元にクラス図を作成してみましょう。

### 3.2.1 設計モデルにクラス図を追加する

まず、「ゲームスコアのクラス図」を追加しましょう。

#### ゲームスコアのクラス図を追加する

- モデルにクラス図を追加する( 図 3.33 )。

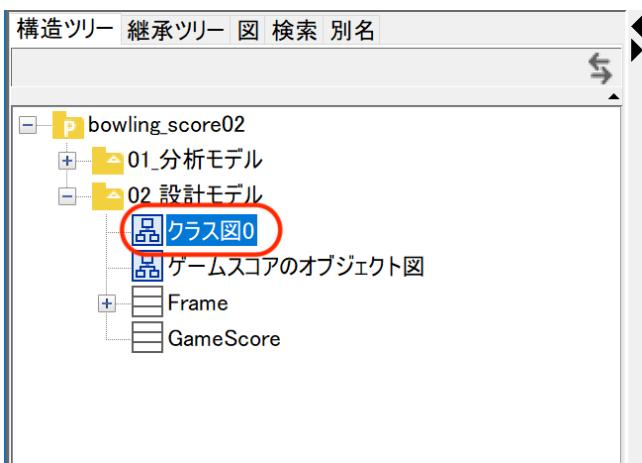


図 3.33 モデルにクラス図を追加する

2. 追加した図を「ゲームスコアのクラス図」とする( 図 3.34 )。

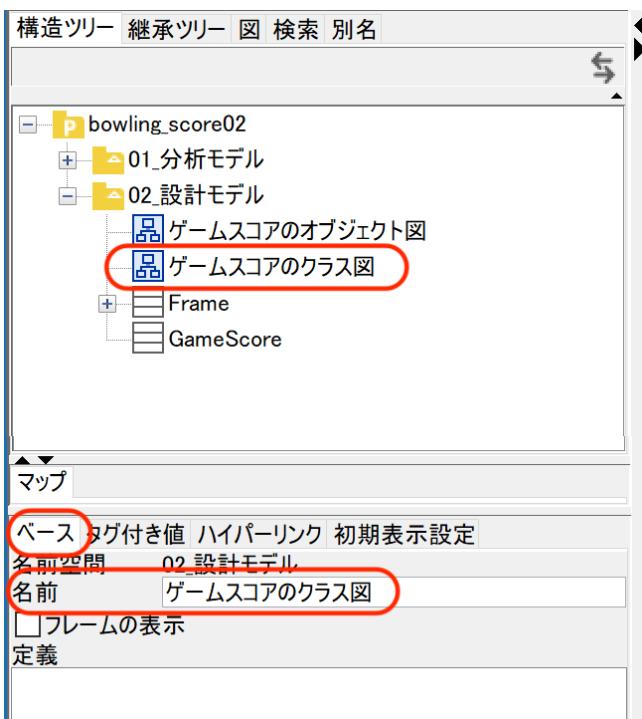


図 3.34 追加した図を「ゲームスコアのクラス図」とした

## 3.2.2 ゲームのクラスを追加する

「構造ツリー」をみると、オブジェクト図を作成したとき登録した「GameScore」クラスや「Frame」クラスが見つかります。これらを1つずつドラッグ&ドロップして、クラス図に追加します( 図 3.35 )。

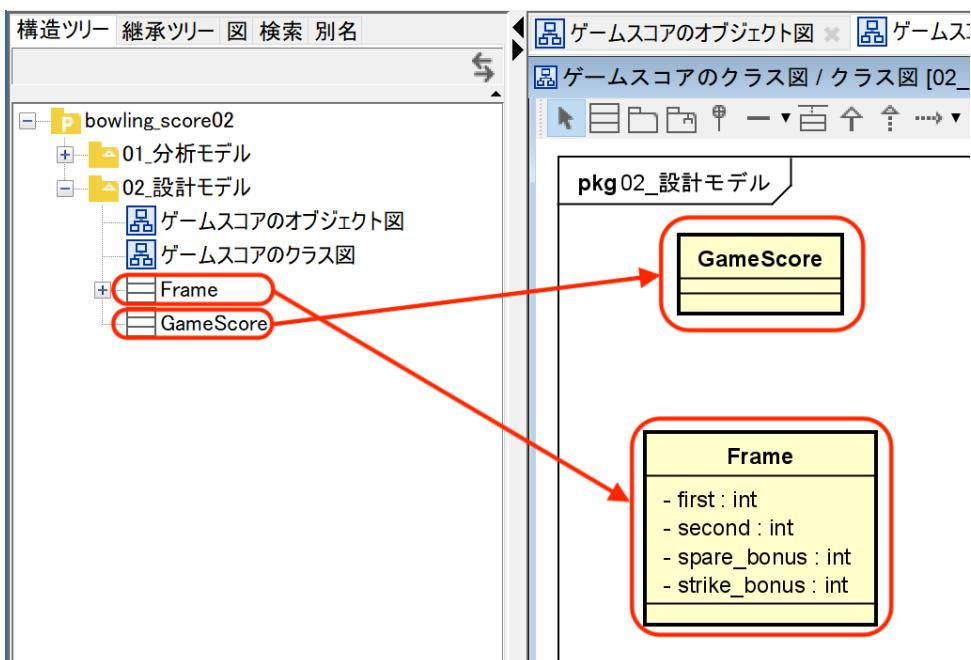


図 3.35 GameScore クラスと Frame クラスをクラス図に追加する

### 3.2.3 クラス間の関連を追加する(1)

前に作成したオブジェクト図( 図 3.32 )を参照して、記載されているリンクを関連として追加します。まず、「 GameScore 」クラスから「 Frame 」クラスへ向かって連を引きましょう。

#### クラス間の関連を追加する

1. パレットから、矢印付きの関連を選択し、「 GameScore 」クラスから「 Frame 」クラスへ向かって関連を引く( 図 3.36 )。

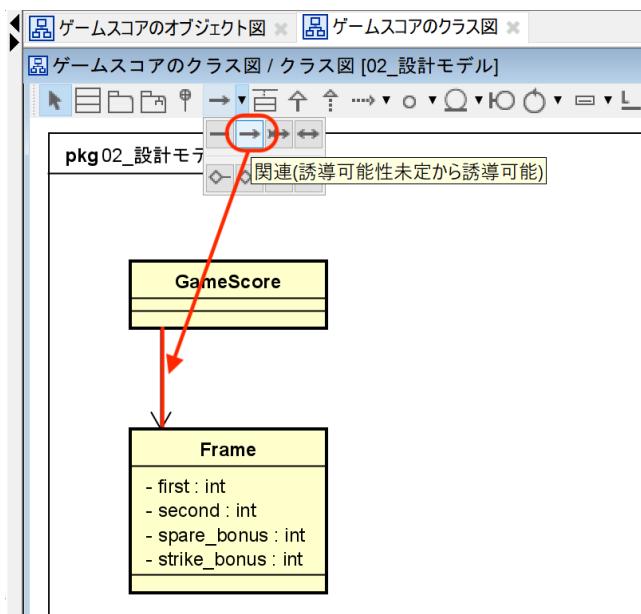


図 3.36 先頭フレームを指す関連「head」を引いた

2. 関連を選択した状態で、プロパティからターゲットが Frame の関連端のタブを開き、名前を「head」とする( 図 3.37 )。

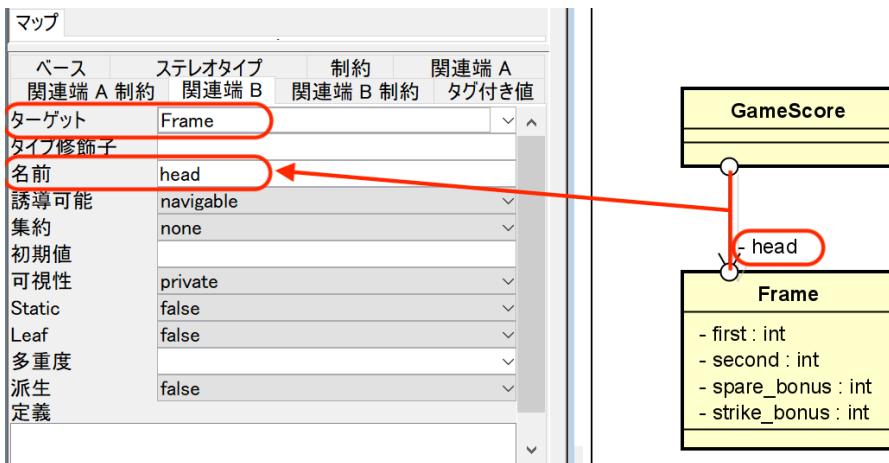


図 3.37 関連端名を「head」とした

3. 「GameScore」クラスからみた「Frame」クラスへの多重度を設定する。

- まだ1投もしていないときはフレームがなく、1投して以降はフレームが複数ある。すなわち、先頭のフレームを指すリンク(関連のインスタンス)は、「つながっていない」か「1つある」かのいずれかになる。
- この事情に見合うよう、関連を選択した状態で、プロパティからターゲットが Frame の関連端のタブを開き、「多重度」を「0..1」に設定する( 図 3.38 )。.

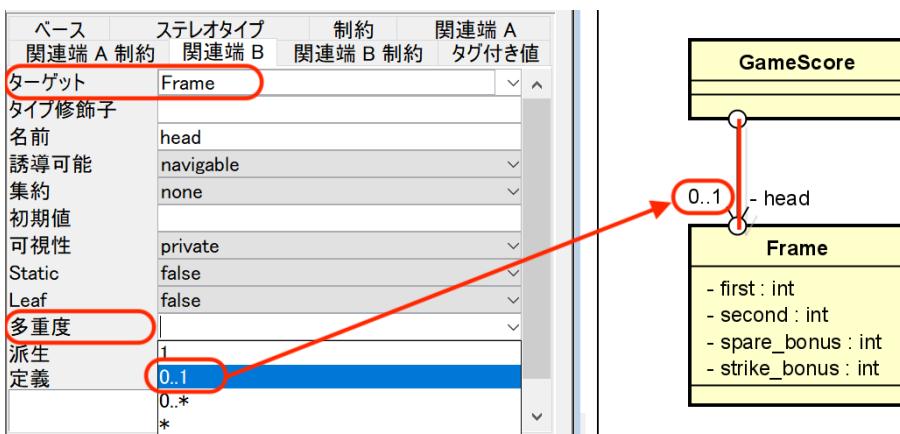


図 3.38 関連の多重度を「0..1」に設定した

4. 「head」と同様、「current」についても関連を引く( 図 3.39 )。

- パレットから、矢印付きの関連を選択し、「GameScore」クラスから「Frame」クラスへ向かって関連を引き、関連端名を付ける。
- 「GameScore」クラスからみると、まだ1投もしていないときはフレームがなく、1投して以降はフレームが複数ある。つまり、現在のフレームを指す関連のインスタンス(リンク)は、「ない」か「1つある」かのいずれかになる。この事情に見合うよう、関連の多重度を設定する。

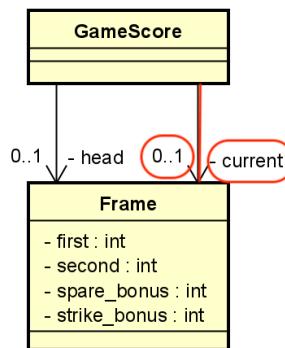


図 3.39 現在のフレームを指す関連「current」を引き、関連端名と多重度を設定した

### 3.2.4 クラス間の関連を追加する(2)

次に、フレーム同士の間に引かれていた「next」「prev」というリンクを関連として引きましょう。もとのリンクは「Frame」クラスのインスタンスから、同じ「Frame」クラスのインスタンスへと引かれていました。このようなリンクを関連に反映するときは、自クラスへつながる関連(自分から自分へ向かう関連)として表します。また、先頭と末尾のフレームは、「nil」とつながっていました。このようなリンクを反映するときは、つながっていない関連があるとみなします。つながっていない場合があるときは、関連の多重度に「0」の場合を含めます。結果として、この部分の多重度は「0..1」となります。

#### フレームからフレームへの関連を引く

1. 「Frame」クラスから「Frame」クラスへ向かって関連を引く。
  - ・パレットから矢印付きの関連を選択する。
  - ・「Frame」クラス上でマウスをクリックして、一度カーソルをクラスから外へ出してクリックし、再び「Frame」クラス上にカーソルを戻してクリックする( 図 3.40 )。
  - ・自分自身へ向かう関連が引けるので、関連の線を整える( 図 3.41 )。

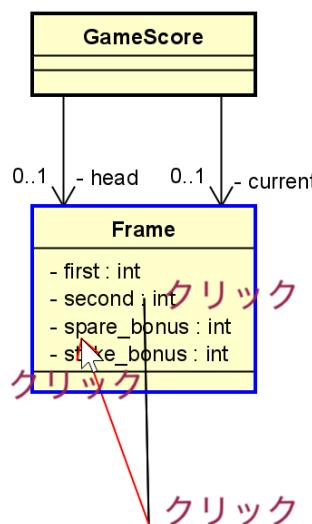


図 3.40 マウスをクラスの上でクリックし、クラスの外でクリックし、再びクラスの上でクリックする

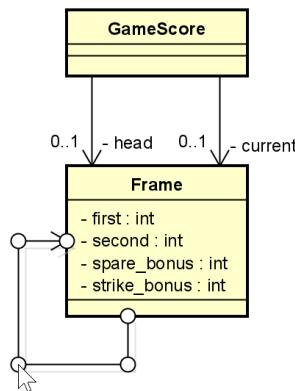


図 3.41 自分自身へ向かう関連が引けたら、関連の線を調整する

2. 関連端名と多重度を設定する

- 関連端名に「**prev**」、多重度に「**0..1**」設定する( 図 3.42 )。

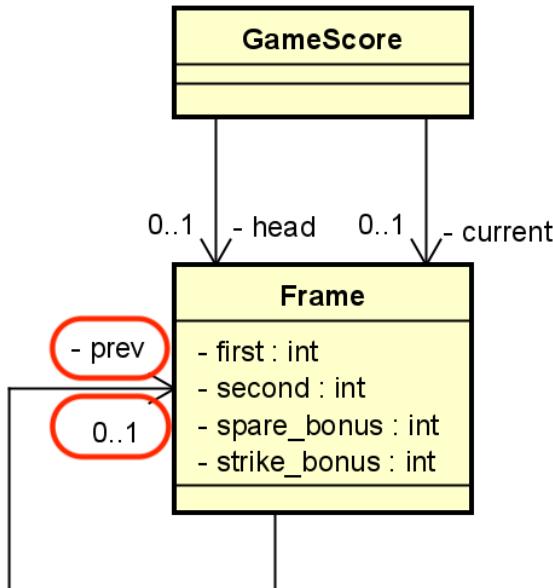


図 3.42 関連端名と多重度を設定した

3. 「**next**」についても関連端名と多重度を設定する( 図 3.43 )。

- 自分自身へ向かう関連を追加する。
- 関連端名を「**next**」、多重度を「**0..1**」とした。

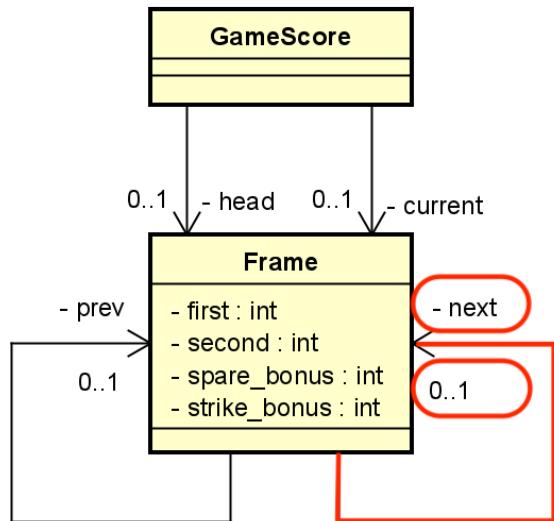


図 3.43 「next」についての関連を追加した

関連「next」と「prev」は、ひとつの双方向の関連として描くこともできます。このときは、ひとつの双方向関連の両端の関連端名を「next」と「prev」とします（図 3.44）。

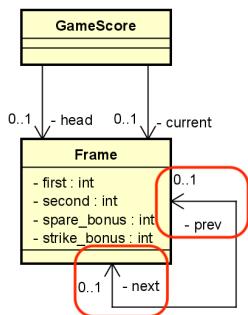


図 3.44 関連「next」と「prev」をひとつの双方向関連で表した場合

## 3.2.5 クラスに操作を追加する

「GameScore」クラスは、ゲームの進行とともに「Frame」クラスのオブジェクトを作成し、スコアを記録していきます。このとき、外部からゲームの進行に応じてピン数を受け取る必要がありますね。そこで、ゲームを進めつつ、ピン数を「GameScore」クラスへ渡す「Game」クラスを追加しましょう（図 3.45）。



図 3.45 Gameクラスを追加した

また、「Game」クラスには、スコアラーの入力を受けつけてゲームを進めるための働きが必要になります。クラスにそのような働きを持たせるときは「操作」を用意します。

ここは、ゲームを進める操作ということで名前を「play」としておきましょう。操作の内容は、振舞いのモデルで検討しましょう。ここでは操作が必要ということまで決めておけばよいでしょう。

#### 「Game」クラスに操作「play」を追加する( 図 3.46 )

1. 「Game」クラスを選択した状態で、プロパティから「操作」タブを選択する。
2. 左下の「+」アイコンをクリックして、操作のエントリを追加する。
3. 操作の名前を編集して「play」にする。
4. クラスの表示も操作「play」が追加される。

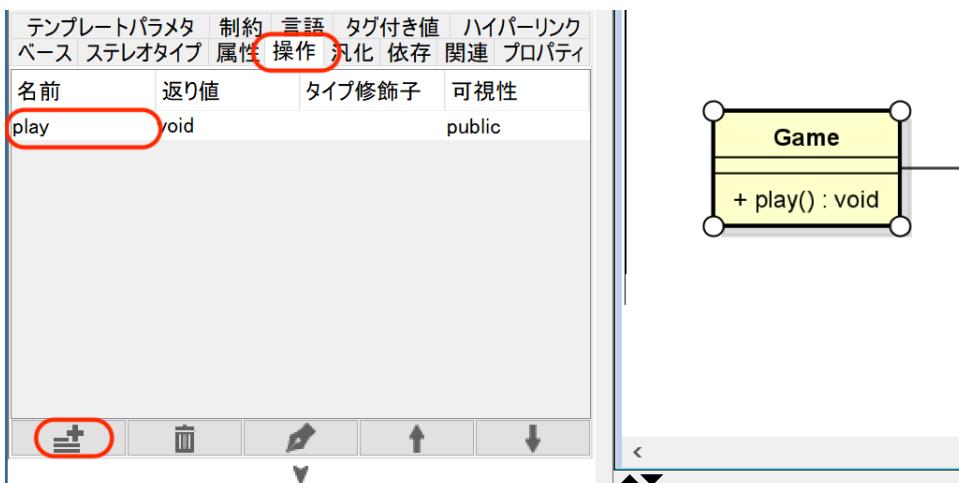


図 3.46 Gameクラスに操作「play」を追加した

また、「GameScore」クラスにも、ゲームの進行とともにピン数を受け取ってスコアを記録する働きが必要になりますね。そこで、「scoring」という操作を用意します( 図 3.47 )。この操作の処理内容も、振舞いのモデルで検討しましょう。「Game」クラスから関連を引きます。関連端名は「game\_score」としました。ゲームがあればスコアは必ず用意するので、多度は「1」に設定しています。

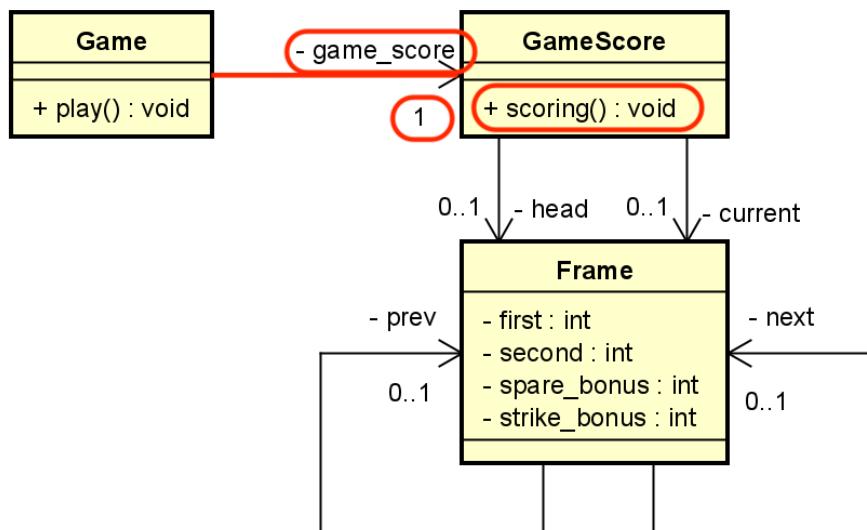


図 3.47 GameScoreクラスに操作「scoring」を追加した

これでクラス図の最初の版はできました。まだ振舞いを検討していませんので、追加した操作は吟味できていません。振舞いを検討するときに、これらの操作についても検討しましょう。

# 4 スコア記録の振る舞いを図にする

スコアの構造を検討できたので、こんどは、ゲームを進めるときの動作をモデル図を使って整理してみましょう。

## 4.1 振る舞いの図を選択する

振る舞いを表す図には「表 4.1」に挙げたようなものがあります。

表 4.1 よく使う振舞いのモデル図

図の種類	説明
シーケンス図	クラスやオブジェクトの間のやり取りを整理するのに向いています。
アクティビティ図	処理の流れや条件による処理の分岐を表すのに向いています。
ステートマシン図	起きるのを待っているできごと(イベント)と、イベントが起きた時の処理(アクション)を使って振る舞いを表すのに向いています。

では、ゲームを進めるときの動作を考えるとき、どの図を使うのがよいでしょうか。たとえば、「Game」クラスと「GameScore」クラスは「GameクラスとGameScoreクラスの間のやり取り」に示すような手順となります。

### GameクラスとGameScoreクラスの間のやり取り

1. システムの利用者は「Game」クラスの操作「play」を呼び出す。
2. 実際のゲームが進行する間、「Game」クラスは次のことを繰り返す。
  - a. 「Game」クラスは、ゲームで投球がある都度、ピン数の入力を受け付ける。
  - b. 「Game」クラスは、「GameScore」クラスの操作「scoring」を呼び出して、入力されたピン数を渡す。
  - c. 「GameScore」クラスは、操作「scoring」を実行してスコアを記録する。

このやり取りを図で表したいなら、シーケンス図が向いているでしょう。ですが、この程度であるなら、必ずしも図を描いて処理の内容を検討するほどではないでしょう。

一方で、「GameScore」クラスの操作「scoring」の動作では、ゲームが進むたびにフレームのインスタンスの追加やボーナスの計算が必要です。また、ボーナスを計算するには、いまどのような状況か(何フレーム目の何投目か)を判断する必要があります。ということは、操作「scoring」の動作では、ゲームの状況を判断するために「状態を維持する」必要がありそうです。そして、ピン数を受け取ったときにスコアを計算するというのは、「できごとが起きたときに何かをする」という考え方で動作を整理する必要があることを意味しています。この考え方につって振る舞いを検討するには「ステートマシン図」を使うのがよさそうです。

## 4.2 ステートマシン図を描く

では、「GameScore」クラスの操作「scoring」の動作を検討するため、ステートマシン図を作成しましょう。

振る舞いの捉え方によって、振る舞いの図がどのようなものになるのかはかわります。この演習では「ピンの入力を待っていてピン数が入力されたらスコアを計算する」という考え方について振る舞いを考えてみましょう。

### 4.2.1 設計モデルにステートマシン図を追加する

まず、「ゲームスコアのステートマシン図」を追加しましょう。追加したいのは「GameScore」クラスの振舞いを表すステートマシン図なので、それがわかるよう「GameScore」クラスに対して追加しましょう。

#### ゲームスコアのステートマシン図を追加する

1. 「GameScore」クラスにステートマシン図を追加する( 図 4.1 )。

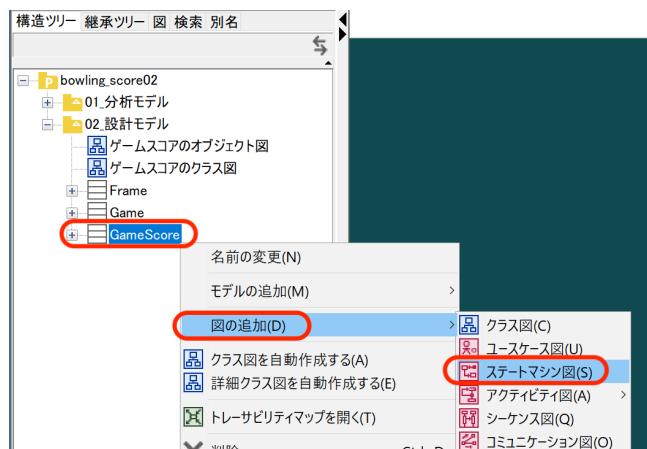


図 4.1 「GameScore」クラスにステートマシン図を追加する

2. 追加した図を「GameScoreのscoringのステートマシン図」とする( 図 4.2 )。

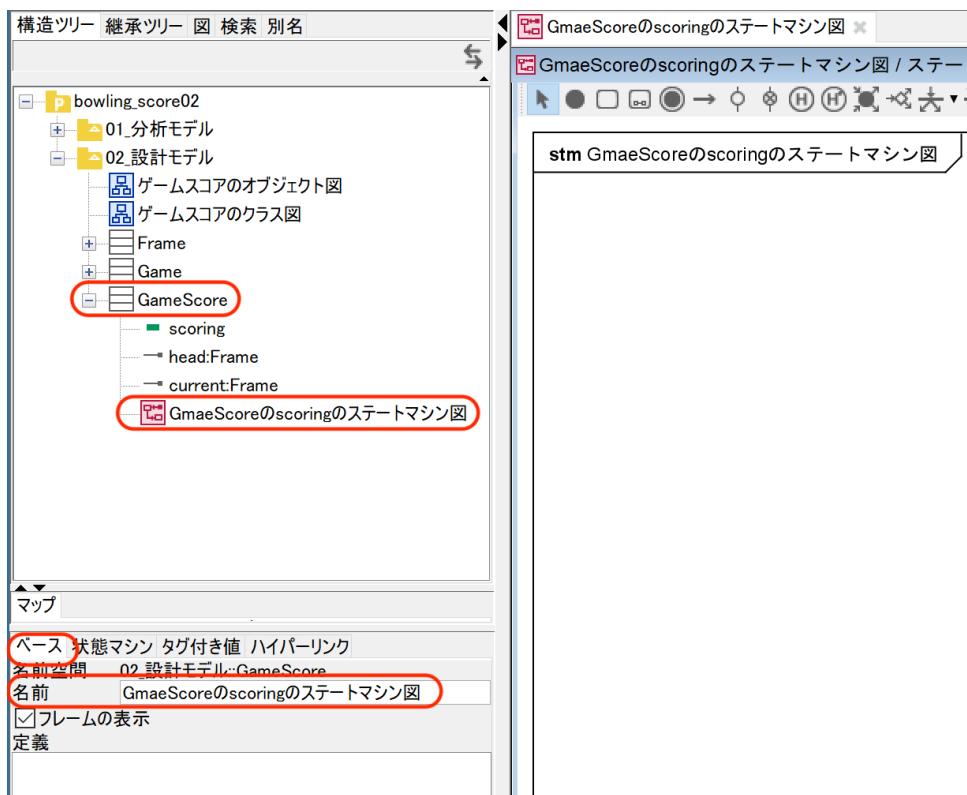


図 4.2 追加した図を「GameScoreのscoringのステートマシン図」とした

## 4.2.2 状態と状態遷移を追加する

### 状態名は後まわしにする



状態を配置すると、状態名をつけたくなります。ですが、状態名をつけるのは後まわしにしましよう。なぜなら、どのような状態なのかを決定する要因は、状態名ではなく、イベントやアクションだからです。そして、イベントやアクションが定まつたら、イベントやアクション（あるいはそれらの元となる仕様上の処理の名前など）を元に状態名を命名します。

ステートマシン図を使って振る舞いを考える時、最初に考えるのは「何が起きるのを待っているのか」です。ユースケース記述「リスト3」を読むと、ゲームを始めた段階では、「1投目のピン数を受け取るのを待っている」と考えればよさそうです。

まず、ステートマシン図に状態と状態遷移を追加しましょう。

### イベント「ピン数を受け取った」を追加する

1. ステートマシン図に状態を2つ追加する。
  - パレットから「状態」のシンボルを選択して、図に配置する( 図 4.3 )。
  - 一方の状態には、最初の状態だとわかるよう「開始疑似状態」をつける。

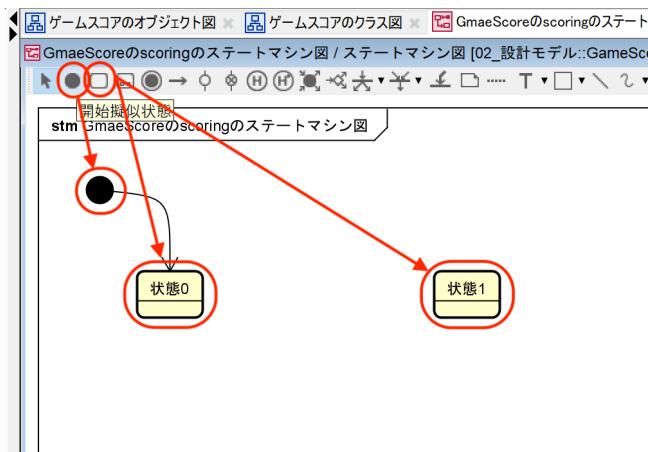


図 4.3 状態と開始疑似状態を追加した

## 2. 状態遷移を追加する。

- パレットから「遷移」のシンボルを選択して、一方の状態にマウスカーソルを移動する。
- 青い枠が現れたら、マウスをクリックし、そのままドラッグする( 図 4.4 )。
- 折り曲げたいときは途中でもマウスをクリックする。
- もう一方の状態の中にマウスを移動して青い枠が現れたらマウスを離すと、「遷移」が引かれる( 図 4.5 )。

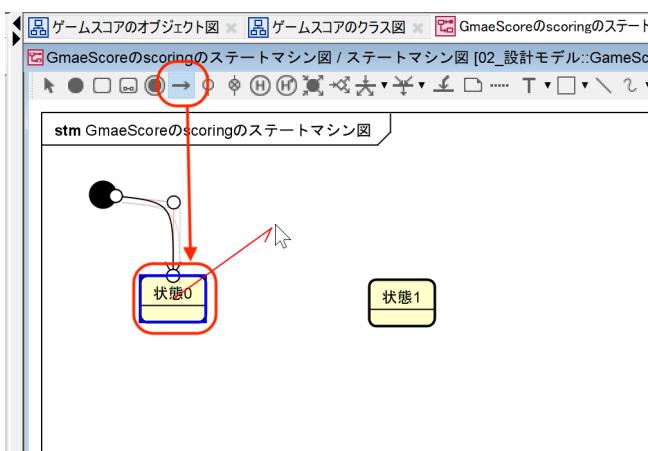


図 4.4 状態と状態の間に遷移を追加する

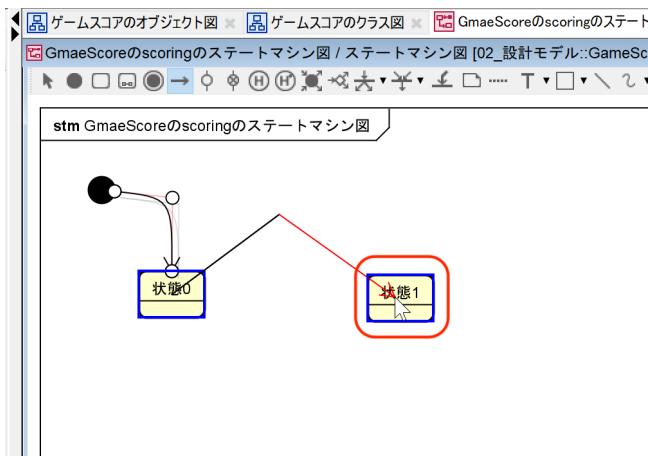


図 4.5 状態と状態の間に遷移を追加した

### 4.2.3 イベントを追加する

状態遷移が追加できたら、最初の状態で起きるのを待っているできごとを考えて、イベントとして追加しましょう。

最初の状態では、投球したピン数を受け取るのを待っていますね。そして、状態が遷移するときは、既に(いままで)受け取ったときになります。そこで、イベントとして表すときには「ピン数を受け取った」のように表します。「受け取る」とすると、まだ受け取るのを待っているところなのか、もう受け取った後なのかはつきりしません。あるいは、「渡す(渡した)」をしてしまうと「GameScore」クラスがピン数を渡すとみなされてしまいます。イベント名を考えるときは、動作の主体やイベントが「起きた」といった時制を意識してみてください。

#### イベント「ピン数を受け取った」を追加する

1. 追加した遷移を選択した状態で、プロパティの「トリガー」を編集して「ピン数を受け取った」とする(図 4.6)。
2. 受け取るピン数がわかるよう、ピン数をパラメータとして追加しておいた。

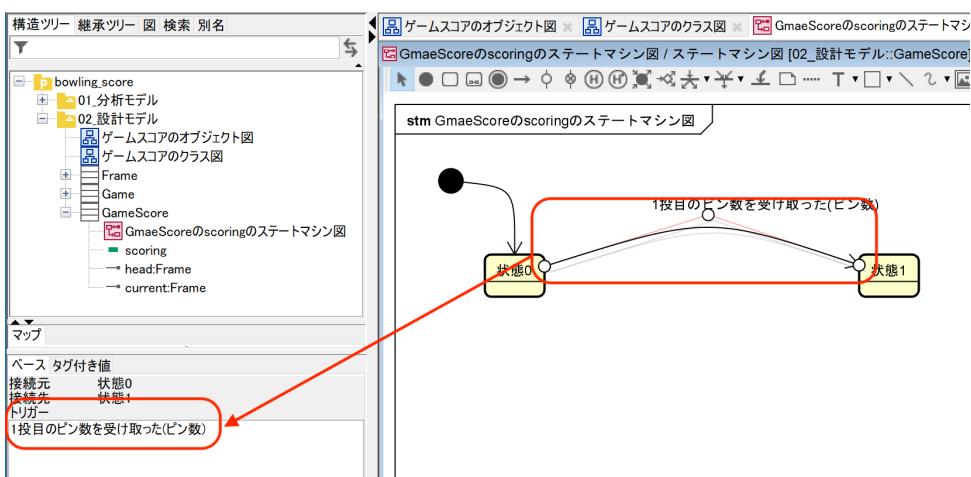


図 4.6 イベント「1投目のピン数を受け取った」を追加した

## 4.2.4 アクションを追加する

次に考えるのは、イベントが起きたときに実行したい処理(アクション)です。1投目のピン数を受け取るのを待っているときにやりたい処理は、ユースケース記述「リスト3」の「現在の状態が、1投目のピン数入力待ちのとき」に書いてある処理です。アクションに記載する処理と次の状態への遷移がわかるよう、抜粋した記述に説明を追加しておきました(「現在の状態が、1投目のピン数入力待ちのとき」の処理(ユースケース記述より抜粋))。

### 「現在の状態が、1投目のピン数入力待ちのとき」の処理(ユースケース記述より抜粋)

1. ゲームスコアに新しいフレームを追加する。(アクションでやりたいこと)
2. 現在のフレームの1投目にピン数を記録する。(アクションでやりたいこと)
3. 1投目がストライクだったときは、現在の状態を次の1投目のピン数入力待ちに変更する。(次の状態への遷移1)
4. ストライクでなかったときは、現在の状態を2投目のピン数入力待ちに変更する。(次の状態への遷移2)

ステートマシン図にアクションを書くときは、このユースケース記述をそのまま書くのではなく、まず「操作に追加してそれをアクションとして呼び出す」方法を原則としましょう。

#### アクションを操作にしてから書く理由



1. その処理に名前をつけることです。名前をつけることで、まとまりのある処理と捉えやすくなります。
2. 再利用の促進です。操作にしておけば、他の場面でも呼び出して使えます。

この演習において、同じ処理が他の状態に現れる場面があるのかはつきりしていませんが、原則に従って作成してみます。

操作の名前は、「1投目のピン数入力待ちのとき」のアクションという意味を込めて「first\_action」としましょう。

### 「1投目のピン数入力待ちのとき」のアクションを追加する

1. 「Frame」クラスに、操作「first\_action」を追加する(図4.7)。
  - 。クラス図を開き、Frame クラスに操作を追加する(操作を追加する手順は省略)。

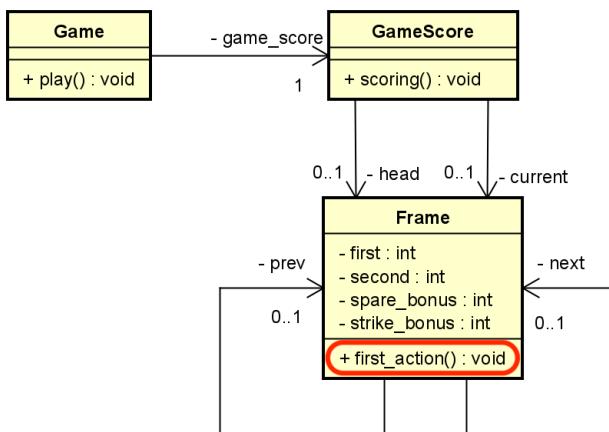


図 4.7 Frameクラスに操作「first\_action」を追加した

2. 追加した操作をアクションに追加する( 図 4.8 )。

- アクションを追加したい状態を選択して、プロパティの「入場/実行/退場」タブを選択する。
- 「入場動作」に「first\_action」と入力する。
- マウスカーソルを図に戻すと、入力が反映される。
- ノートを追加して、アクション詳細を書いておいた。

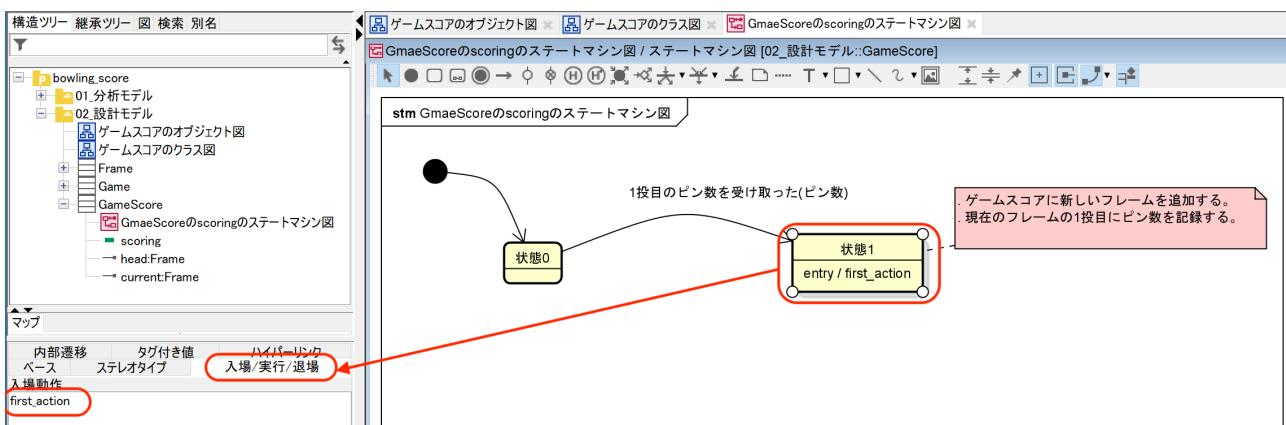


図 4.8 アクション「first\_action」を状態に追加した

そして、「「現在の状態が、1投目のピン数入力待ちのとき」の処理(ユースケース記述より抜粋)」の後半部分、次の状態への遷移を追加します。記述をみると状態遷移は2つあります。まず、考えやすいストライクでなかったときを考えていきましょう。このときは、2投目のピン数を受け取るのを待つことになりますので、そのための状態遷移とイベントを追加します。

2投目のピン数入力待ちに遷移する状態遷移を追加する( 図 4.9 )。

1. 新しい状態を追加する。
2. 追加した状態へ向かう状態遷移を追加する。

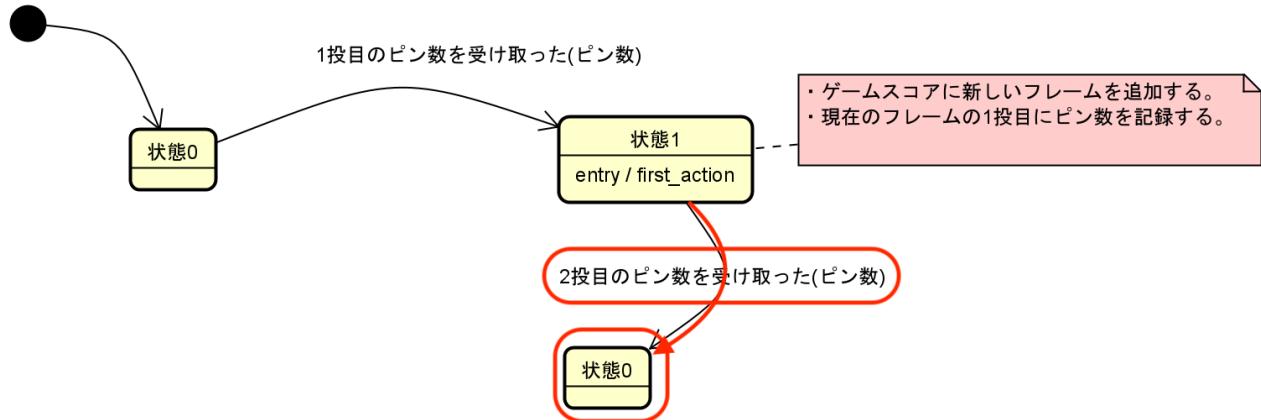


図 4.9 2投目のピン数入力待ちに遷移する状態遷移を追加した

## 4.2.5 ストライクのときの状態遷移を追加する

では、ストライクだったときはどうしたらよいでしょうか。このときは、「1投目のピン数入力待ちのとき」のアクションを実行した上で、次のフレームの1投目のピン数を受け取るのを待つことになります。つまり、前の状態に戻るということです。この状態遷移を追加してみましょう。

ストライクだったときの状態遷移を追加する(図 4.10)。

1. 最初の状態への状態遷移を新しい状態を追加する。
2. 追加した状態へ向かう状態遷移を追加する。

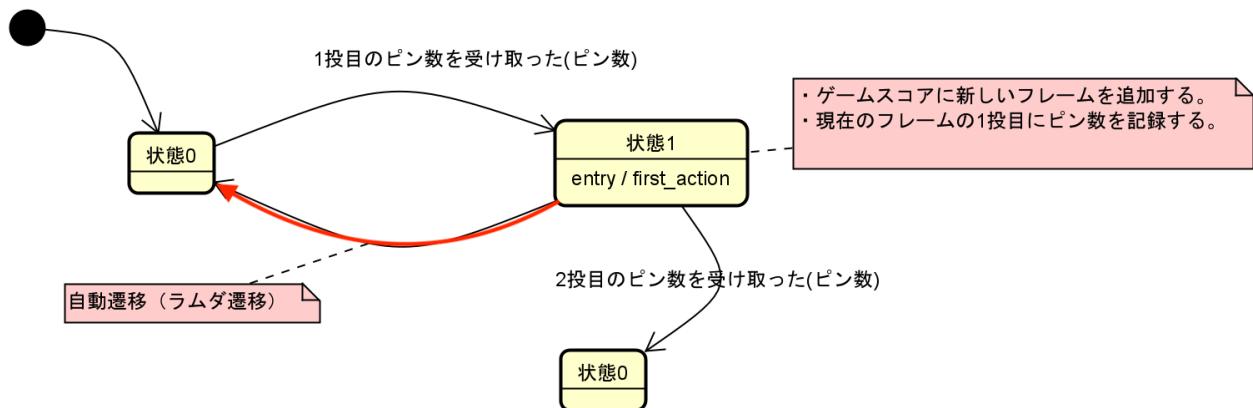


図 4.10 次の1投目のピン数入力待ちに遷移する状態遷移を追加した(期待通りに動作しない)

この状態遷移には、待っているイベントがありません。イベントが書いていない遷移は、遷移元の状態におけるアクションが実行された後、イベントが発生していない状態が遷移するという意味になります。このような状態遷移を「自動遷移(ラムダ遷移)」といいます。

しかし、この自動遷移があると、2投目のピン数を受け取るのを待っているにもかかわらず、すぐこちらの状態遷移で遷移してしまいますね。これは期待している動作ではありません。つまり、ストライクのときとそうでないときは、状態処理は同じですが区別する必要があります。区別するということは、2投目を待つ場合と、ストライクで次のフレーム

を進む場合は別の状態と考えるわけです。

同じイベントを受け取ったときに、条件によって遷移先を変えたい場合は、イベントにガード条件を追加します。ここでは、「ストライク」の場合と「ストライクでない」場合です。

### ストライクかどうかで状態遷移を区別する

1. ストライクのときの状態を追加する( 図 4.11 )。

- 新しい状態を追加する。
- 最初の状態から追加した状態へ状態遷移を引く。
- 追加した状態から最初の状態へ状態遷移を引く。

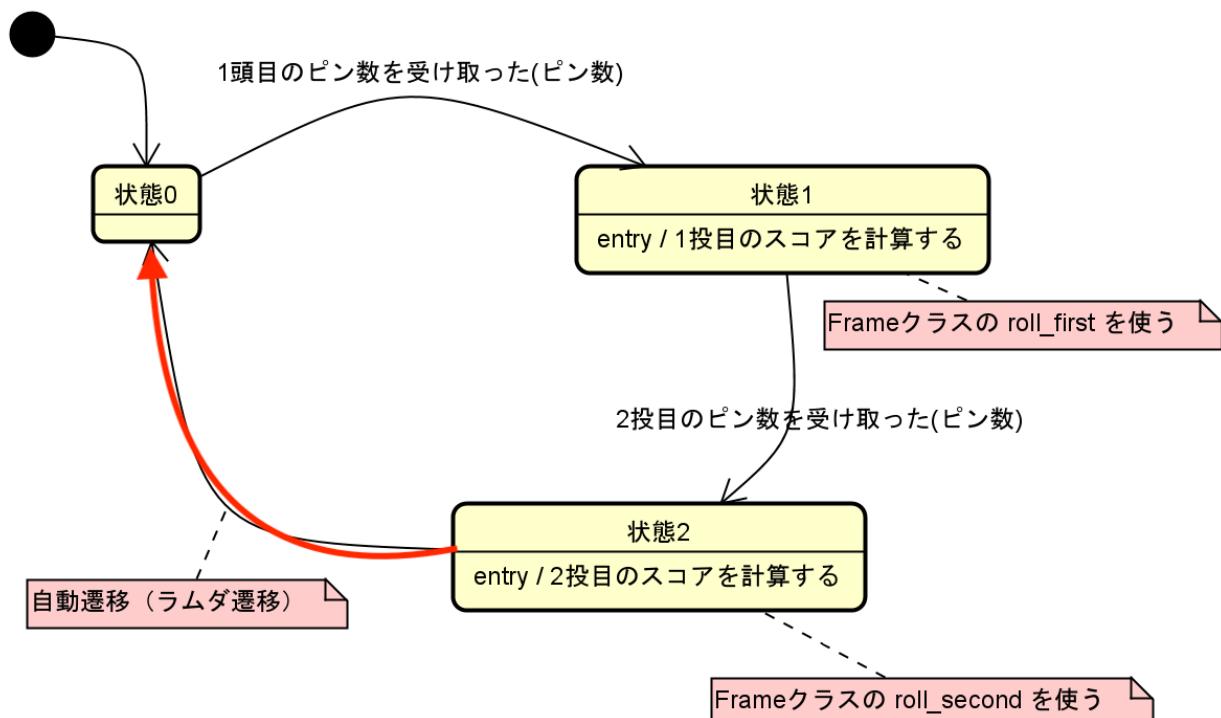


図 4.11 ストライクのときのために状態を追加した

## 4.2.6 イベントとアクションの追加を繰り返す

引き続き、ユースケース記述「リスト3」を参照して、イベントとアクションを追加します。

1投目のピン数を受け取った状態では、起きるのを待っているできごとは「2投目のピン数を受け取る」ことです。イベントとしては「2投目のピン数を受け取った」になります。

ふたたび、操作と状態と状態遷移を追加して、このイベントを追加します。追加する操作は、「roll\_second」とします。

### アクション「2投目のスコアを計算する」を追加する

1. 「Frame」クラスに、操作「roll\_second」を追加する( <<class21→> )。

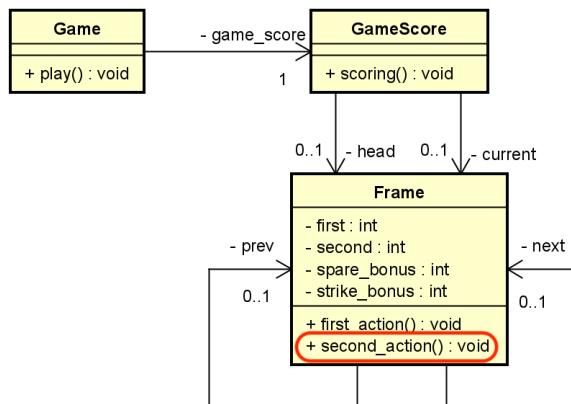


図 4.12 Frameクラスに操作「roll\_second」を追加した

2. イベントとアクションを追加する( 図 4.9 )。

- 状態と状態遷移を追加する。
- 状態遷移に、イベント「2投目のピン数を受け取った」を追加する。
- 状態に、アクション「2投目のスコアを計算する」を追加する。
- ノートを追加して「Frameクラスの roll\_second を使う」と書いておく。

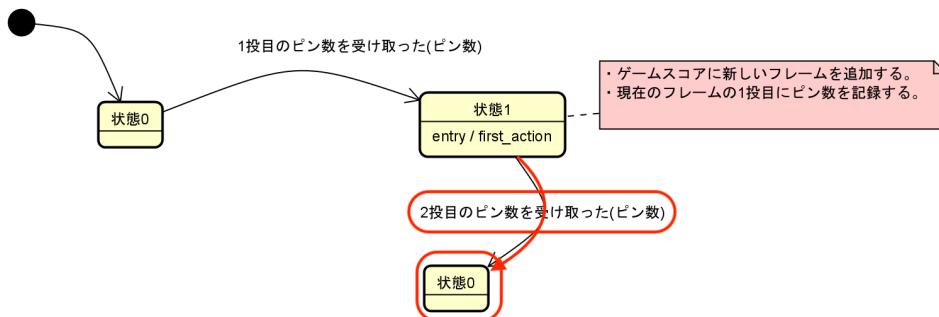


図 4.13 アクション「2投目のスコアを計算する」を追加した

そして、もし、ストライクやスペアの処理を勘案しないのであれば、この後は最初の状態に戻ればよいですね。その場合、ステートマシン図は「<<stm08-->>」のようになるでしょう。

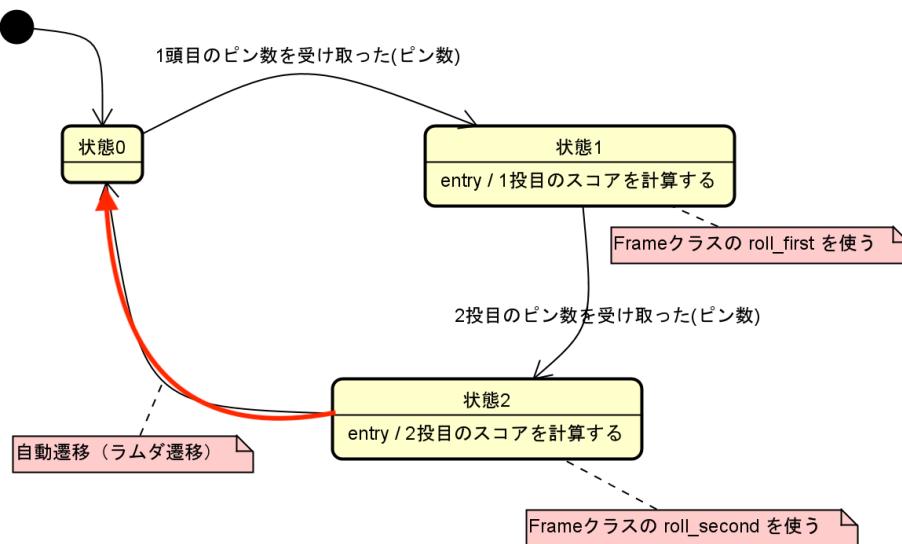


図 4.14 最初の状態に戻る自動遷移を追加した

この状態遷移にはイベントがありません。このような状態遷移を「自動遷移（ラムダ遷移）」といいます。自動遷移の場合、前の状態のアクションが実行された後、イベントが発生していくなくても状態が遷移します。

これで、毎フレーム必ず2投するのを延々繰り返すように動作するステートマシン図ができました。

## 4.2.7 ストライクの処理を追加する

「1投目のピン数を受け取った」などの処理の詳細は、ユースケース記述「リスト3」の「1投目のスコアを計算する」に書いてあります。これを「[1投目のスコアを計算する処理の詳細](#)」に抜粋しておきます。

### 1投目のスコアを計算する処理の詳細

1. ゲームスコアに新しいフレームを追加する。
2. 現在のフレームの1投目にピン数を記録する。
3. 1投目がストライクだったときは、現在の状態を次の1投目のピン数入力待ちに変更する。
4. ストライクでなかったときは、現在の状態を2投目のピン数入力待ちに変更する。

「[図 4.11](#)」には、1投目で10ピン倒してストライクになったときの処理が反映できていません。反映するには、ステートマシン図をどのように修正すればよいでしょうか。

まず、1投目のピン数を受け取ったら1投目のスコアを計算するのは変わりありません。そして、受け取ったピン数が10ピンであったときは、2投目のピン数を受け取るのを待たずに次のフレームへ進みます。そのために、たとえば「[図 4.15](#)」のような状態遷移を追加してみたらどうでしょうか。

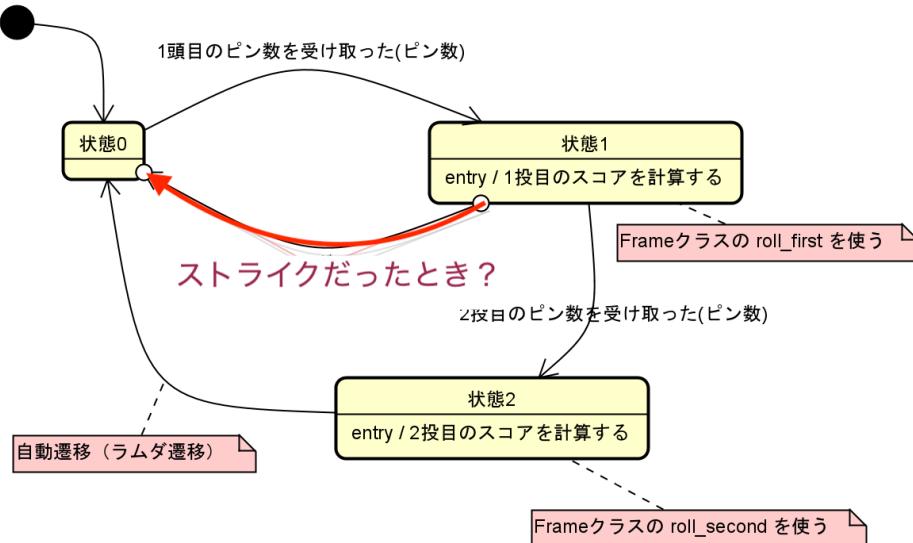


図 4.15 ストライクのときの状態遷移の検討(1)(期待通りに動作しない)

一見、これでなんとかなりそうに思えます。ところが、ここに自動遷移を書いてしまうと、1投目のスコアを計算した後は必ずこの自動遷移へ向かいます。ここでは、2投目のピン数を受け取るのも待たなくてはなりませんが、いつもそれよりも先に自動遷移してしまうでしょう。つまり、ここに自動遷移を書いてしまうと、2投目のピン数を受け取るイベントが来るのを待たなくなってしまうのです。

それでは、ストライクで次のフレームへ進むために、別の状態を追加してみます(図 4.16)。

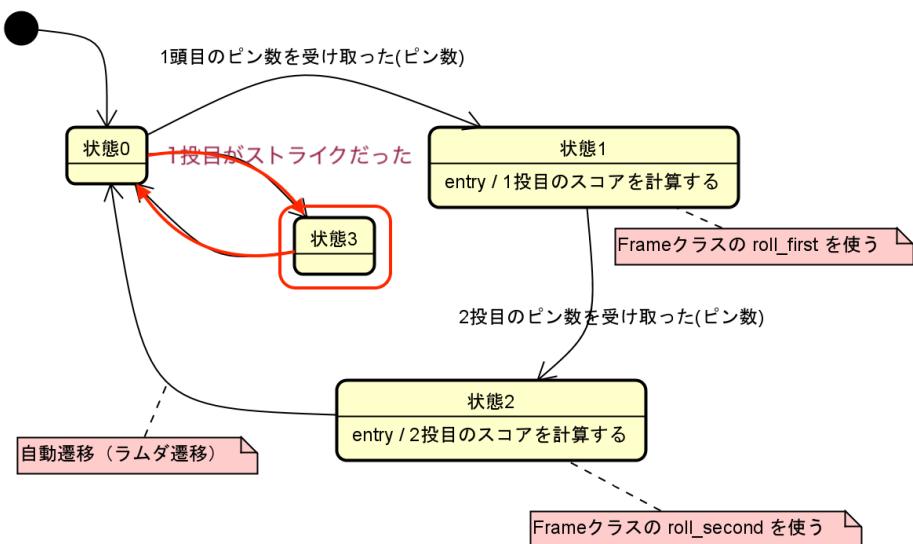


図 4.16 ストライクのときの状態遷移の検討(2)(あいまいさが生じる)

これで、「2投目のピン数を受け取るのを待つ」ところには影響がなくなりました。自動遷移によって次のフレームの1投目を待つところへ戻るのも問題ないでしょう。しかし、この図では「1投目のピン数を受け取った」という同じイベントで、2つの遷移先ができてしまっています。このように、同じイベントを待つ2つの状態遷移があると、どちらへ遷移するのかあいまいになってしまいます。このままでは、どのように動作するのか決められません。

この問題を避けるには、1投目のピン数を受け取ったときに、それがストライクなのか調べて区別すればよいでしょう。そこで、イベントにはイベントが起きたときに評価する条件を追加できるようになっています。この条件のことを「ガード

条件」と呼びます。



ガード条件は、イベントを待っているときではなく、イベントが起きたときに評価されることを注意してください。

まず、ストライクか調べて真偽を返す操作を追加しましょう。追加する操作の名前は「strike?」としましょう。



Rubyでは、真偽を返す操作(Rubyではメソッドと呼びます)の末尾に「?」をつける習慣があります。

### ストライクのときの状態と状態遷移を追加する

1. 「Frame」クラスに、操作「strike?」を追加する( 図 4.17 )。

- 。「strike?」を選択した状態で、プロパティのベースタブの「返り値」のプルダウンメニューから「boolean」を選択する。

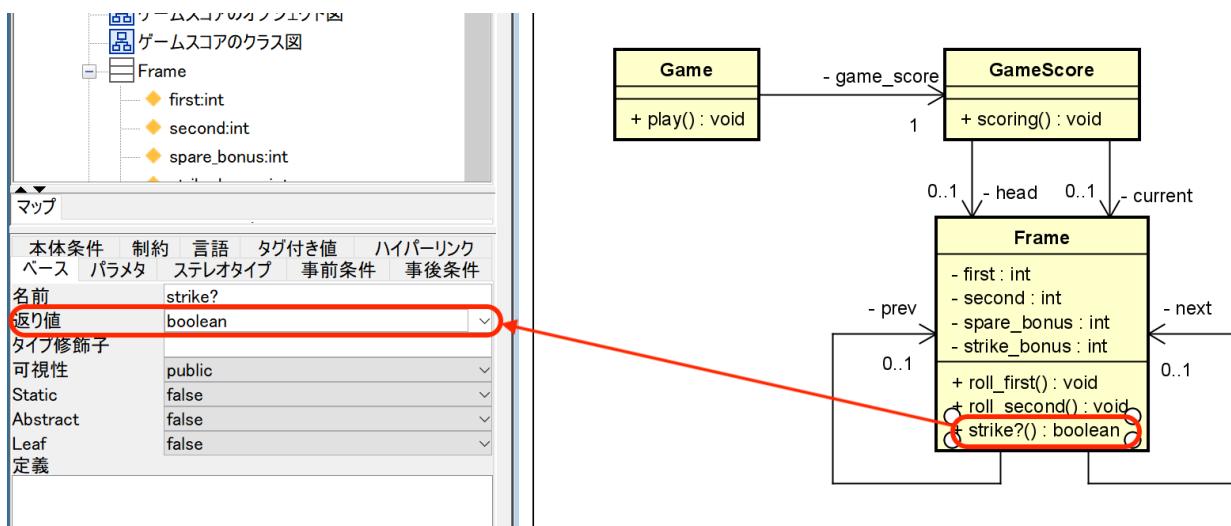


図 4.17 Frameクラスに操作「strike?」を追加した

2. イベントにガード条件を追加する( 図 4.18 )。

- 。イベント「1投目のピン数を受け取った」を選択した状態で、プロパティの「ガード」欄に「ストライクではなかった」と書く。
- 。マウスカーソルを図に戻すと、図上のイベントにガード条件が追加される。
- 。状態遷移にノートを追加して「Frameクラスの strike? を使う」と書いておく。
- 。前からあった状態遷移の方イベントは「1投目のピン数を受け取った[ストライクではない]」に変更する。
- 。追加した状態のアクションに「1投目のスコアを計算する」を追加する。

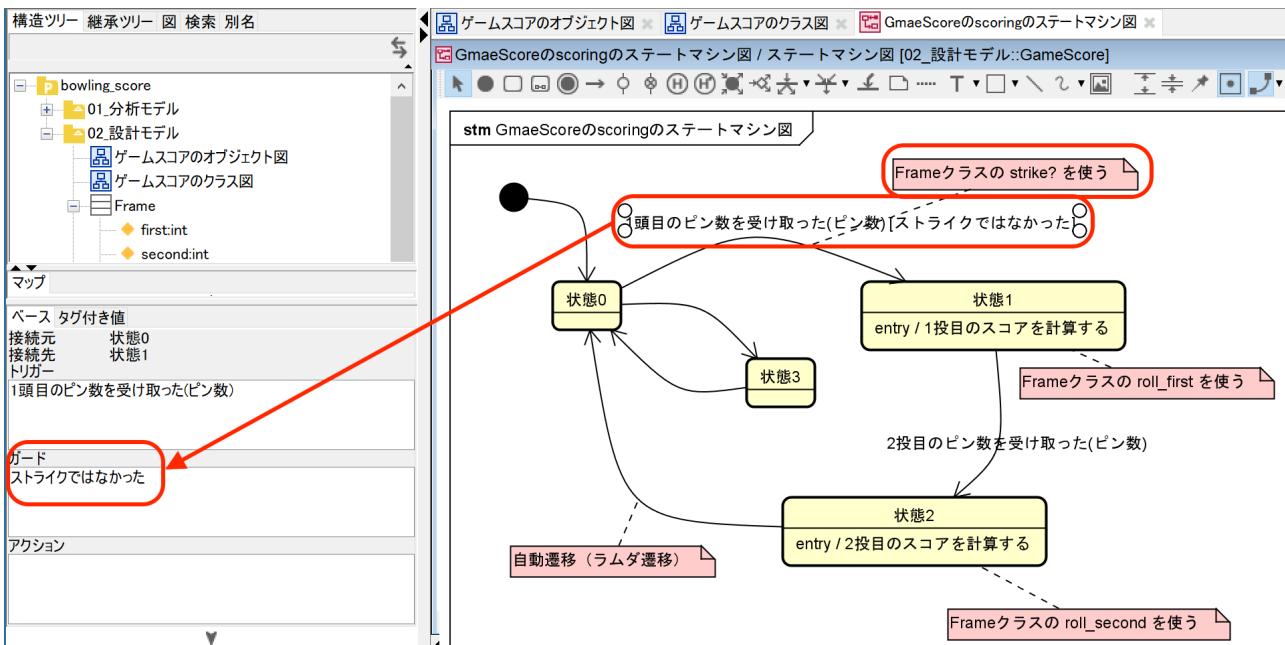


図 4.18 イベントにガード条件を追加した

3. ストライクのときの状態を状態遷移を追加する( 図 4.19 )。

- （まだ追加していなかった場合）ストライクのときのための状態と状態遷移を追加する。
- 追加した状態遷移に、イベント「1投目のピン数を受け取った[ストライク]」を追加する。
- 追加した状態のアクションに「1投目のスコアを計算する」を追加する。

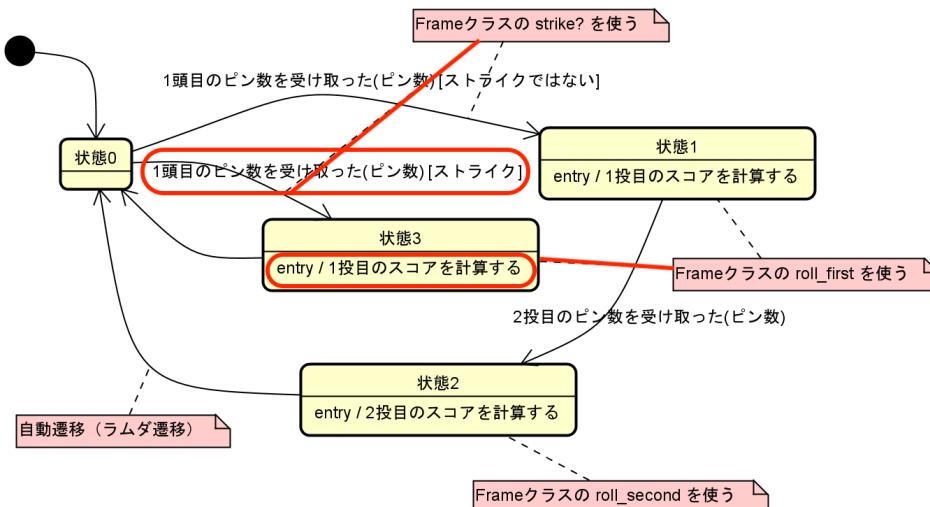


図 4.19 ガード条件を使ってストライクのときの状態と状態遷移を追加した

これで、同じイベントが起きても、ガード条件の評価結果が異なれば別の状態へ遷移できるようになりました。

## 4.2.8 サービスフレームの処理を追加する

最終の第10フレームは、3投目を投球できます（サービスフレームと呼ぶこともあるそうです）。この処理に対応するにはどうしたらよいでしょうか。

まず、最終フレームの3投目のピン数を保持するために、3投できるような Frameクラスを用意しましょう。これは、Frame クラスを継承すれば作成できそうです( 図 4.20 )。

そして、最終フレームを追加するのは、通常のフレームと同じように操作 add\_frame に任せましょう。その代わり、現在のフレームが何フレーム目なのかわかるよう、GameScoreクラスに作成しているフレームの数を覚えておく属性「size」を追加します。

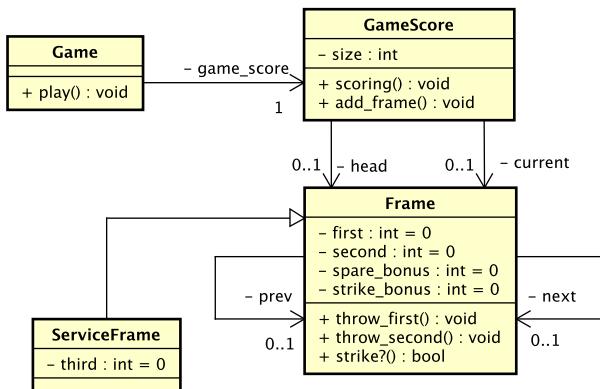


図 4.20 「ServiceFrame」クラスを追加し、GameScoreクラスに属性「size」を追加した

## 4.2.9 スペアの判定を追加する

1投目が10ピンでない場合で、2投目で残りのピンをすべて倒した場合は「スペア」になります。Frameクラスに、スペアかどうか判定する操作「spare?」を追加しておきましょう( 図 4.21 )。

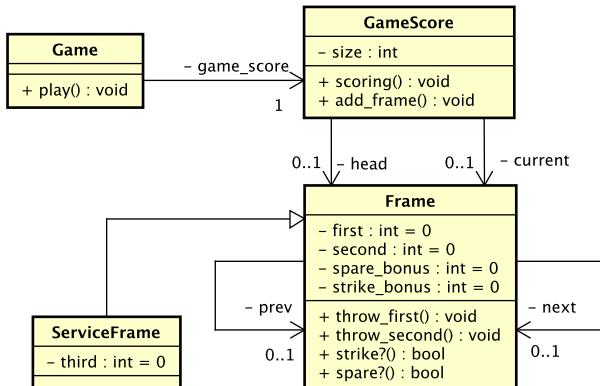


図 4.21 Frameクラスに操作「spare?」を追加する

## 4.2.10 状態名をつける

[stm06\_svg]について、イベントとアクションが決まったので、これらを元に状態名をつけます。状態名をつけるときは「<<state\_naming\_rules>」のような原則を使ってつけます。

### 状態名をつけるときの原則

- 待っているイベントやそのイベントを期待する状況を使って「～待ち」とする

2. 実行中のアクティビティやそのアクティビティが関連する状況を使って「～中」とする

3. 最後の状態は「～待ち」や「～中」とはつけず、「終了」「完了」「到着」などとする

状態名は、待っているイベントのうち最も期待しているイベント元に「～待ち」とつけます。ところが、イベント名そのものを使うと、複数の状態が同じイベントを待っていると状態名が重複してしまいます。そこで、仕様やユースケース記述を参照して、そのイベントが起きるときの状況を表わしている名前を探して状態名にします。たとえば「ピン数受け取り待ち」などとなるでしょう。

もし、イベントの発生を待っている間、継続して実行している処理があれば、おそらく「do アクティビティ」の処理として書いてあるでしょう。この処理を元に「～中」とつける方法もあります。この場合も、アクションそのものではなく、仕様やユースケース記述からそのアクションを実行している状況を示している名前を参照してつけます。たとえば「2投目投球中」などとなるでしょう。



状態名に「待機中」を使う場合は、要注意です。対象業務の業務用語として「待機中」が定義されている場合は、そのときを表す名前として使うこともできるでしょう。これは、業務の用語であるなら、その「待機」が何を待っているのかについても業務として明らかなことが多いからです。ですが、それ以外の場合は、再考してみた方がよいでしょう。たいていの場合、待っているということそのものよりも「待っているイベント(や、そのイベントを期待する業務上の状況)は何か」の方が重要です。

それでは、状態名をつけてみましょう。1投目については、同じイベントに対する別の状態遷移と状態を追加します(図 4.22)。

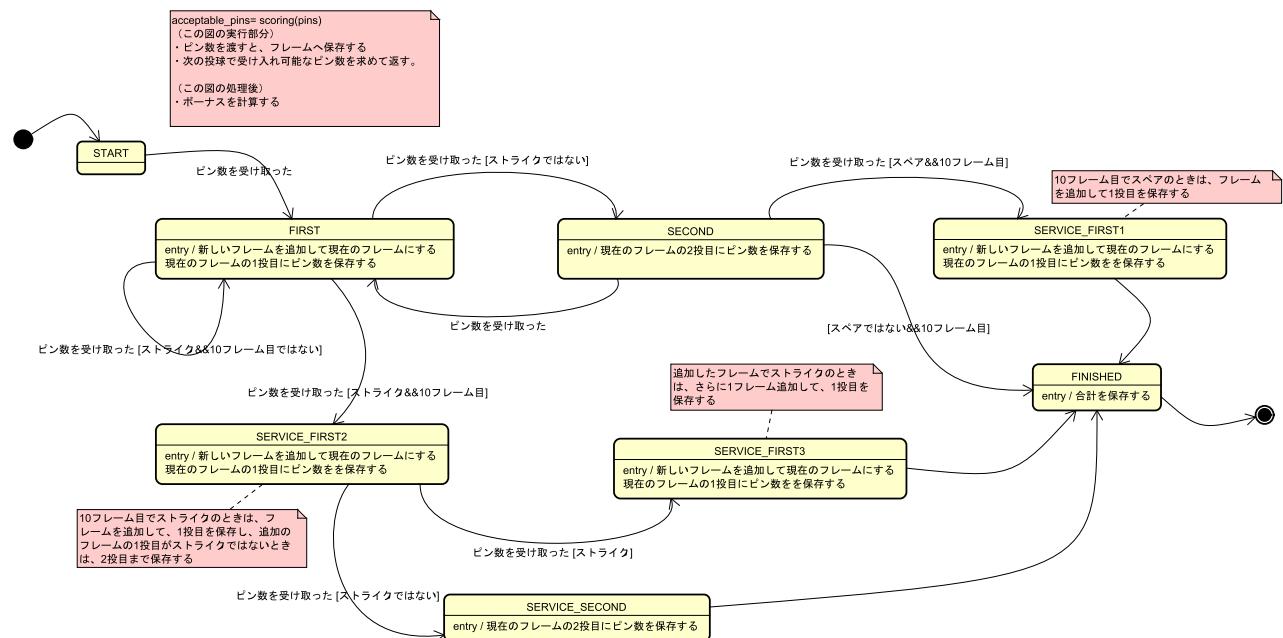


図 4.22 状態名をつけた

# 5まとめ

このチュートリアルでは、ボーリングスコアのモデルを作ることを通して、ソフトウェアの開発にモデルを使うことについて学びました。また、UMLが、そのようなときに役立つ記法であることも実感できたのではないでしょうか。ソフトウェアの開発にモデルを活用するときは、記法やツールも欠かせません。このチュートリアルでは、モデル図を描くのにastah\*を使いました。ですが、このチュートリアルをやったみなさんは、記法やツールがあればモデルが作れるわけではないことに気づいたのではないでしょうか。

まず、このチュートリアルでは、ボーリングスコアのモデルをいきなり作るのではなく、「誰のどんな問題か」を確認することから始めました。現実的には、ボーリングスコアのような自明なよく知られた問題であれば、そのようなことをいちいち議論する必要はないでしょう。しかし、システム開発案件の多くは、どのように対処すればよいのかわかつていないう題を抱えています(だからこそ、そこが開発案件になるわけですよね)。このチュートリアルでは、そのようなときに問題を表すモデルを作るのと同じように、ボーリングスコアの記録に関わる領域をモデルで表すところから始めました。

みんなのなかには、UMLの考え方や記法に慣れていないだけで、モデル化して考えることには慣れている人もいるでしょう。そのような人は、こんどは、自分たちがすでにモデル化している図や文書をUMLを使って表してみるとよいでしょう。

あるいは、すでに作りたいもの(あるいは解決したい課題)はわかってきてているものの、それがどのような課題でどのように解決すればよいのかといったことを、うまく表せずに困っている人もいるでしょう。そのような人は、みんなの課題(を含む現状)をモデル図で表してみるとよいでしょう。これを「As Isのモデル」と呼びます。そして、浮き彫りになった問題を、そのモデルを出発点として解消したモデルを作ります。これを「To Beのモデル」と呼びます。課題を解決したシステムは、To Be のモデルを元に設計、実装します。このような方法で、問題解決にモデルを活用してみることにチャレンジしてみてください。

みんながこれまで以上にモデルを活用できるようになることを期待しています。



# 6 他の図、他の機能など



| 他の図、astah\* の他の機能のことを書く。

# 付録A: 関連資料、ワークシート等

- ・サンプルモデルとプログラム
  - sample01.zip

# 用語集

## 変換ルール

前工程のモデルから後工程モデル、設計のモデルからコードなど、異なる工程の間には、表記や意味に違いが生じる。このとき、それぞれの工程の成果物の要素（モデルの要素）同士を対応づけると、成果物のつながりが確保できる。そのような対応づけを「変換ルール」と呼ぶ。

## マジックナンバー（magic number）

直接数値のまま使っていて、値の意味がわからない識別子番号や数値定数のこと。`const` 修飾子やマクロ定義を使って名前をつけることで、わかりやすくする対処方法がよく用いられる。

## ドメイン（domain, 問題領域）

あるシステムにおいて取り扱う必要があることがらが複数の領域に渡る場合、領域ごとに使われることばや隣接する領域によってそのシステムを分けたほうがよい。その分けた領域のことを「ドメイン（問題領域）」と呼ぶ。領域に分けることを「ドメイン分割（domain separation）」と呼ぶ。一般に、システムやサブシステムは複数のドメインによって構成される。たとえば、自動搬送ロボットを使った運搬システムの場合、運搬業務としては荷物を運ぶ手順、自動搬送ロボットには運搬に使う装置としての走行動作や外部との操作や通信、自動搬送ロボットを構成するデバイスではモーターを動かすというように、それぞれが使うことばが異なっており、これらは異なるドメインとみなせる。

## ドメイン分割（domain separation）

「ドメイン」を参照。

## サブシステム（subsystem）

あるシステムにおいて、開発単位が異なる（別のプログラムに分かれる、別途開発のライブラリなど）や、操作者や通信が介在することで動作が区切られるような場合、分けたそれをそのシステムの「サブシステム」と呼ぶ。サブシステムに分けることを「サブシステム分割（subsystem separation）」と呼ぶ。一般に、サブシステムは複数のドメインによって構成される。たとえば、PCによる管制局と自動搬送ロボットで構成される自動搬送システムを考えると、管制局、自動搬送ロボットともに運搬業務のことばを扱う運搬業務のドメインを持つが、開発単位が異なり通信も介在するのでそれぞれサブシステムとみなせる。

## サブシステム分割（subsystem separation）

「サブシステム」を参照。

## ロバストネス図（robustness diagram）

ロバストネス分析に用いる記法。クラスについて「バウンダリ」「エンティティ」「コントロール」の3種類のステレオタイプと専用のアイコンを与え、ここにアクターを加えたクラス図を描くことで構成要素を探しやすくなる。

## ロバストネス分析（robustness analysis）

システムの構成要素は「バウンダリ」「エンティティ」「コントロール」の3種類に大別できると考え、これらを探すことでシステムの構成要素を抽出しようとする分析方法。記法として「ロバストネス図」を用いる。図の要素の接続関係には、アクターはバウンダリとしか接続が持てない、エンティティはコントロールを介してしか他とつながることができないなどの制約があり、この制約によって構成要素を探しやすくなっている。

### オブジェクト図(object diagram)

システムの構成要素を、実際に登場する具体的な要素(オブジェクト)を使って表した図。インスタンス図とも呼ばれる。

### インスタンス図(instance diagram)

「オブジェクト図」を参照。

# 参考文献

- [UMLSPEC] OMG 統一モデリング言語(UNIFIED MODELING LANGUAGE).
  - <https://www.omg.org/spec/UML/>
- [SYSML] OMG System Modeling Language (SysML).
  - <http://www.omg.org/spec/SysML/>
- [mathgirl] 数学ガール.
  - <https://www.hyuki.com/girl>
- [robustness01] ワークブック形式で学ぶUMLオブジェクトモデリング.
  - ローゼンバーグ, スコット. ソフトバンク. 2002.
- [robustness02] ユースケース駆動開発実践ガイド.
  - ローゼンバーグ, ステファン. 翔泳社. 2007.
- [rulebook] ボーリング競技規則.
  - <https://jbc-iwate.com/service/message/201012101.pdf>

# モデルを使って ソフトウェアを開発しよう

## astah\* を使った UMLチュートリアル

---

発行日 : 2022-01-17

バージョン : pdf\_0011

作成者 : 株式会社チェンジビジョン

本書の内容に関する質問等がありましたら、作成者までお知らせください。