## アドバンスト・ファイティング・ファンタジー第２版用の専用ダイスをみてみよう

**Faceless**

　オンラインでTRPGをプレイするときに、何が必要でしょうか？ マップ、立ち絵にコマ……え～と。そうそう、大体のゲームでダイスは必須ですよね？ プレイ中にダイスコマンド(1D6やFF7など、ダイスを振るときの命令ですね)を打ったことがある人は沢山いるでしょう。打つと結果が出てくるのを便利に思ったことがある人も沢山いると思います。

　このダイスがどんな風に動いているか調べてみたことがありますか？ 調べたことのある人はあまり多くないですよね(笑)

　ここでは、アドバンスト・ファイティング・ファンタジー第２版(以下、AFF2e)用の専用ダイスのプログラムが、どのように動いているかを解説してみようと思います。コンピュータ・プログラミングに興味がある方以外には、プログラム的な細かい話をしても意味がないと思います。ここではプログラミング言語の解説は極力控えて、プログラムが何をやっているかだけに集中して解説していこうと思います。「へ～こんな風に動いているんだ」「案外単純だな」と思っていただけると幸いです。

　ちなみに専用ダイスのプログラムは「Ruby」というコンピュータ言語で書かれています。「Ruby」はなかなか面白い特徴を持っているので、興味を持たれた方はWebや書籍で是非調べてみてください。

---

　まずは、プログラムの全体をお見せします。当たり前ですが、中身が判る必要はありません。

```
# frozen_stri# frozen_string_literal: true

module BCDice
 module GameSystem
  class AFF2e < Base
   # ゲームシステムの識別子
   ID = 'AFF2e'

   # ゲームシステム名
   NAME = 'ADVANCED FIGHTING FANTASY 2nd Edition'

   # ゲームシステム名の読みがな
   SORT_KEY = 'あとはんすとふあいていんくふあんたしい2'

   # ダイスボットの使い方
   HELP_MESSAGE = <<~MESSAGETEXT
     対抗なしロール\tFF{目標値}+{補正}
     対抗ロール\tFR{能力値}+{補正}
     武器ロール\tFD[2,3,3,3,3,3,4]+{補正}
     防具ロール\tFD[0,0,0,0,1+1,1+1,2+2]+{補正}
```

※このコードは ぱるたさん(@parco_opaai) によって書かれたもので、BCDice有志によってメンテナンスされています。
https://github.com/bcdice/BCDice/pull/237

```
MESSAGETEXT

# ダイスボットで使用するコマンドを配列で列挙する
register_prefix('FF.+', 'FR.+', 'FD.+')

def explicit_sign(i)
 format('%+d', i)
end

def eval_term(term)
 value = 0
 term.scan(/[+-]?\d+/) do |fact|
  value += fact.to_i
 end
 value
end

def parentheses(str)
 '(' + str + ')'
end

def successful_or_failed(total, diff)
 case total
 when 2
  diff <= 1 ? '成功（大成功ではない）' : '大成功！'
 when 12
  diff >= 12 ? '失敗（大失敗ではない）' : '大失敗！'
 else
  total <= diff ? '成功' : '失敗'
 end
end

def critical(total)
 case total
 when 2
  'ファンブル！'
 when 12
  '強打！'
 end
end
```

```
def clamp(i, min, max)
 if i < min
  min
 elsif i > max
  max
 else
  i
 end
end

def eval_game_system_specific_command(command)
 case command
 when /\AFF/
  # 対抗なしロール
  # '成功' or '失敗' を出力する
  #
  md = Regexp.last_match
  term = md.post_match

  # 目標値
  diff = eval_term(term)

  dice_command = "2D6<=#{diff}"
  dice_list = @randomizer.roll_barabara(2, 6)
  total = dice_list.sum()
  dice_str = dice_list.join(",")
  expr = "#{total}[#{dice_str}]"
  succ = successful_or_failed(total, diff)
  sequence = [parentheses(dice_command), expr, succ]
 when /\AFR/
  # 対抗ロール
  # 値を出力する
  #
  md = Regexp.last_match
  term = md.post_match

  # 補正値
  corr = eval_term(term)
```

```
    dice_command = "2D6#{explicit_sign corr}"
    dice_list = @randomizer.roll_barabara(2, 6)
    total = dice_list.sum()
    dice_str = dice_list.join(",")
    expr = "#{total}[#{dice_str}]#{explicit_sign corr}"
    crit = critical(total)
    sequence = [parentheses(dice_command), expr, crit, total + corr].compact
   when /\AFD/
    # 武器防具ロール
    # ダメージを出力する
    #
    md = Regexp.last_match
    term = md.post_match
    md = /\A\[(.+)\]/.match(term)
    unless md
     return 'ダメージスロットは必須です。'
    end

    term = md.post_match
    damage_slots = md[1].split(',').map { |t| eval_term(t) }
    if damage_slots.size != 7
     return 'ダメージスロットの長さに誤りがあります。'
    end

    # 補正値
    corr = eval_term(term)

    dice_command = "1D6#{explicit_sign corr}"
    total = @randomizer.roll_once(6)
    expr = "#{total}#{explicit_sign corr}"
    slot_number = clamp(total + corr, 1, 7)
    damage = damage_slots[slot_number - 1]
    sequence = [parentheses(dice_command), expr, total + corr, "#{damage}ダメージ"]
   end

   result = sequence.join(' ＞ ')
   return result
  end
 end
end
end
```

案外短いと思ったでしょうか？ それとも「うわっ！長い」と思ったかな？ この中に「対抗ロール」「対抗なしロール」「武器ロール」「防具ロール」と４つの機能が書かれています。(武器ロールと防具ロールはダイスコマンドとしては、ひとつに纏められています)

さて、この手のコンピュータ・プログラムには読み方にコツがあります。なるべく小さな「ひと固まりごと」に何をやっているのか掴むのが定石です。一気に全体を把握するのは大変ですが、なるべく小さなパーツに分解してしまえば理解しやすいですからね。この辺は、ルールブックを読むときと似ているかもしれません。分量的には、ずっと少ないですけどね！

プログラムは基本的に上から下に順番に処理されるので、上から順番に読んでいきましょうか。

```
# frozen_string_literal: true

module BCDice
 module GameSystem
  class AFF2e < Base
   # ゲームシステムの識別子
   ID = 'AFF2e'

   # ゲームシステム名
   NAME = 'ADVANCED FIGHTING FANTASY 2nd Edition'

   # ゲームシステム名の読みがな
   SORT_KEY = 'あとはんすとふあいていんくふあんたしい2'

   # ダイスボットの使い方
   HELP_MESSAGE = <<~MESSAGETEXT
     対抗なしロール\tFF{目標値}+{補正}
     対抗ロール\tFR{能力値}+{補正}
     武器ロール\tFD[2,3,3,3,3,3,4]+{補正}
     防具ロール\tFD[0,0,0,0,1+1,1+1,2+2]+{補正}
   MESSAGETEXT

   # ダイスボットで使用するコマンドを配列で列挙する
   register_prefix('FF.+', 'FR.+', 'FD.+')
```

まずは、この辺までにしましょう。「#」で始まる行はコメントです、プログラムを読むときに判り易いように随時書きこんであります。このあたりのプログラムはコメントを見るとなんとなく何が言いたいか判りますよね。コメントはプログラムの動作に影響しません。

例外は最初の行の

```
# frozen_string_literal: true
```

です。

コメントなのに、プログラムの動作に関係があるので「マジック・コメント」なんて言ったりします。「文字列リテラル」というものを固定するかい？ →はい　という設定を行っています。なんのこっちゃと首をひねりつつ「そうすると便利なことがあるんだろうな……」と思った貴方は鋭い！ 良くあるミスを防ぐためのありがたい呪文なのです。

また、プログラムの左側が妙な具合に空間が開いていますね。実は「module」から「end」までで、一区切りなのです。moduleだけ書いてendを忘れるとプログラムが動かなくなってしまうので、対応が視覚的に判るように右に「字下げ」しているのです。

---

さて、続きを解説していきましょう。

```
module BCDice　(←ダイスエンジン「BCDice」のモジュール(部品)です)
 module GameSystem　(←その中でも「GameSystem」のモジュールです)
(ここで、「自分が何者か」を明らかにしています)

  class AFF2e < Base　(←AFF2eに基本のダイスの機能を一通り与えます)
   # ゲームシステムの識別子
   ID = 'AFF2e'
   (↑「=」は代入です。IDに'AFF2e'を入れます。これでどのゲームシステムかを区別しています。)

   # ゲームシステム名
   NAME = 'ADVANCED FIGHTING FANTASY 2nd Edition'
   (↑ゲームシステム名をココフォリア等で何と表示するかを指定しています)

   # ゲームシステム名の読みがな
   SORT_KEY = 'あとはんすとふあいていんくふあんたしい2'
   (↑読み方順に並べるときの読み方の登録です。濁点などを省くことになっています)

   # ダイスボットの使い方
   HELP_MESSAGE = <<~MESSAGETEXT
     対抗なしロール\tFF{目標値}+{補正}
     対抗ロール\tFR{能力値}+{補正}
     武器ロール\tFD[2,3,3,3,3,3,4]+{補正}
     防具ロール\tFD[0,0,0,0,1+1,1+1,2+2]+{補正}
   MESSAGETEXT
   (↑「ダイスの使い方」で表示するヘルプ・メッセージの設定をしています。
     「\t」はタブ文字の意味です)
```

```
# ダイスボットで使用するコマンドを配列で列挙する
register_prefix('FF.+', 'FR.+', 'FD.+')
```
(↑オンセツールのチャットでダイスが誤爆しないように、使うダイスコマンドを設定しています)

この辺は設定でしたね。設定が散らばっていると修正や記入漏れの確認が大変なので、一か所にまとめてあるのが判ります。

---

次のセクションは「便利な部品」のプログラムが書かれています。moduleと同じようにdefの次に部品の名前が書かれています。後に出てくるプログラムで、部品を名前で呼ぶと部品が処理してくれる仕組みになっています。部品も「def」～なんとかかんとか～「end」なので字下げしていますね。

仮に「部品集」としておきましょう。

```
def explicit_sign(i)
 format('%+d', i)
end
```
(↑数字に「+-」の記号を付ける部品です)

```
def eval_term(term)
 value = 0
 term.scan(/[+-]?\d+/) do |fact|
  value += fact.to_i
 end
 value
end
```
(↑足し算、引き算を計算する部品です)

```
def parentheses(str)
 '(' + str + ')'
end
```
(↑文字の並び「文字列」を括弧で囲む部品です)

```
def successful_or_failed(total, diff)
 case total
 when 2
  diff <= 1 ? '成功（大成功ではない）' : '大成功！'
 when 12
  diff >= 12 ? '失敗（大失敗ではない）' : '大失敗！'
 else
  total <= diff ? '成功' : '失敗'
 end
end
```
(↑達成値と難易度を比べて、成功・失敗・大成功・大失敗の表示をする部品です)

```
def critical(total)
 case total
 when 2
  'ファンブル！'
 when 12
  '強打！'
 end
end
(↑達成値をみて、ファンブル・強打を表示する部品です)

def clamp(i, min, max)
 if i < min
  min
 elsif i > max
  max
 else
  i
 end
end
(↑最大値、最小値の範囲に数値をまとめる部品です。ダメージスロットの決定に使います)
```

便利な部品集を最初に書いてあるのは、最初は「ふんふん、こんな処理が必要になるのね」と流し読みしておくくらいにして、後になって出てきてから「え～と、あそこに書いてあったな」と振り返るのに便利なやり方です。最初は本当にちょっとしたことをやる部品から書いて、だんだんと成功失敗の判別やクリティカル・ファンブルの判定と順序良く出てくるあたりに、このプログラムを組んだ人の性格がちょっと出ていますよね。

---

いよいよダイスコマンドに入ります。このAFF2e専用ダイスはコマンドの処理をcase～when～endで区分けしているので、whenごとに区切ってみていきましょう。

```
def eval_game_system_specific_command(command)
 (↑ゲーム専用のコマンドを書いていく特別な部品)
case command
when /\AFF/   (←FFコマンド)
 # 対抗なしロール
 # '成功' or '失敗' を出力する
 #
(↑コメントで、このコマンドがどういう処理を行うかを簡潔に説明していますね)
```

```
md = Regexp.last_match
term = md.post_match
(↑コマンドから目標値の式を取り出す)

# 目標値
diff = eval_term(term)
(↑目標値を計算する。さっき出た部品集に中身が書いてある)

dice_command = "2D6<=#{diff}"
(↑「2D6<=目標値」の文章(文字列)を記録)

dice_list = @randomizer.roll_barabara(2, 6)
(↑ダイスを2d6振って)

total = dice_list.sum()
(↑合計値を記録する)

dice_str = dice_list.join(",")
(↑さっき振った2d6の出目も記録)

expr = "#{total}[#{dice_str}]"
(↑「達成値[出目]」の文字列を作る)

succ = successful_or_failed(total, diff)
(↑成功失敗の結果を調べる)

sequence = [parentheses(dice_command), expr, succ]
(↑表示する結果を順番に並べて記録しておく)
```

このFFコマンドが基本になって、他のコマンドも同じように処理されていきます。どのコマンドも「コマンドの判定」→「パラメータの取り出し」→「修正値の計算」→「出力する文字列の生成と記録」という順番で整然と流れていきます。

```
when /\AFR/    (←FRコマンド)
 # 対抗ロール
 # 値を出力する
 #
 md = Regexp.last_match
 term = md.post_match
 (↑補正値の式を引っ張り出して)
```

```
# 補正値
corr = eval_term(term)
(↑補正値の計算をする)

dice_command = "2D6#{explicit_sign corr}"
(↑ダイスコマンドを「2D6+補正値」で記録)

dice_list = @randomizer.roll_barabara(2, 6)
(↑2D6を振って)

total = dice_list.sum()
dice_str = dice_list.join(",")
(↑合計値と出目を記録する)

expr = "#{total}[#{dice_str}]#{explicit_sign corr}"
(↑「目標値[ダイス目]補正値」で記録)

crit = critical(total)
(↑クリティカル・ファンブルを判定)

sequence = [parentheses(dice_command), expr, crit, total + corr].compact
(↑表示する結果を順番に並べて記録しておく)
```

FFとFRを比べると、目標値と補正値が入れ替わり表示がちょっと変わった感じですが、概ね似たような処理ですね。同じような処理を「どこまで纏めて、どこから分けるか」もプログラムを組む人の腕の見せ所です。ここでは、多少似ていてもコマンドごとに処理を分けることに決定した訳ですね。

---

次のコマンドをみてみましょう。

```
when /\AFD/　(←FDコマンド)
 # 武器防具ロール
 # ダメージを出力する
 #
 md = Regexp.last_match
 term = md.post_match
 md = /\A\[(.+)\]/.match(term)
 (↑ダメージスロットを取り出す)

 unless md
  return 'ダメージスロットは必須です。'
 end
 (↑もし無かったらメッセージを出して終了)
```

```
(ここからはダメージスロットがある場合)
term = md.post_match
damage_slots = md[1].split(',').map { |t| eval_term(t) }
(↑ダメージスロットのデータを取り出す)

if damage_slots.size != 7
  return 'ダメージスロットの長さに誤りがあります。'
end
(↑ダメージスロットのデータの長さを調べる。７以外ならメッセージを出して終了)

(ここからは、ダメージスロットが正しく存在している場合)
# 補正値
corr = eval_term(term)
(↑補正値を計算する)

dice_command = "1D6#{explicit_sign corr}"
(↑ダイスコマンドを「1D6+補正値」にする)

total = @randomizer.roll_once(6)
expr = "#{total}#{explicit_sign corr}"
(↑1d6を振って、「結果+補正値」で記録)

slot_number = clamp(total + corr, 1, 7)
(↑何番目のダメージスロットを使うか計算する)

damage = damage_slots[slot_number - 1]
(↑ダメージスロットからダメージを取り出す)

sequence = [parentheses(dice_command), expr, total + corr, "#{damage}ダメージ"]
(↑表示する結果を順番に並べて記録しておく)

end
(↑case～when～endを閉じるend)
```

FDコマンドはちょっと複雑でした。ダメージスロットの記述ミスがありうるので、確認してエラーメッセージを出す処理が追加されていますね。ダメージスロットを忘れた時と、長さが違う(途中で抜けたり重複したり)時でメッセージが違うのも芸が細かいです。

---

コマンド処理の最期は、記録してある結果を「＞」で繋げて出力して終了です。

```
    result = sequence.join(' > ')
    return result
  end
```

　プログラムの最期は、classやmoduleの相棒のendが並んでいますね。

```
  end
 end
end
```

　如何でしたか？

　処理している内容そのものは「なあんだ、ルール通りじゃないか」と思ったのではないでしょうか？ そうなのです、ルールを過不足なく解釈してRubyという言語で書き直したものが、専用ダイスのプログラムという訳です。逆にいうと、ルール処理が判らないとプログラムは組めません。

　専用ダイスの要望を出すと、ルールについて根掘り葉掘り聞かれることがあると思いますが、プログラムを組み上げるのに必要な情報という訳なのです。　ルールの解説をするつもりで、なるべく詳細かつ具体的に説明すると喜ばれます。

　なお、専用ダイスのコマンド追加などの要望はBCdiceのdiscodeサーバなどにて受け付けていますので、お気軽にどうぞ！

(oldbookillustrations)