基般

ゲームを通してプログラミングの基礎を学ぼう（前編）
― Racket で学ぶプログラミング―

対馬かなえ（国立情報学研究所）

Racket で始める楽しいゲームプログラミング

今，「ゲーム」といえば，ゲーム機やスマホやパソコンでプレイするコンピュータゲームのことです．「ゲームを楽しむ」とは，世の中の数多くのゲームから自分が好きになれそうなものを選び，プレイを楽しむことです．プレイするだけではなく，自分の創作したゲームで，ほかの誰か，知らない誰かを楽しませ，感動してもらうというゲームの「楽しみ方」ができるようになったら，どんなに素晴らしいでしょう？　とはいえ，高速グラフィクスが美麗な市販のアクションゲームなどを見ていると，なかなか「自分にも作れるのでは?」という気持ちにはなれないかもしれません．でも，そのようなゲームも，プログラムの内容は，意外と単純な繰り返しでできていたりします．本稿と次号の後編では，未完成の簡単なゲームを完成させることで，プログラミングを楽しく学び，身につけ，「自分でできる!」と実感していただくことを目指します．最後に，これから「自分ならでは」のゲームを作っていくためのコースの入口までご案内します．何はともあれ，「プログラミングって思ったより簡単だし，結構楽しい」と思っていただければ，幸いです．

今回使用するのは，「Racket」というプログラミング言語です．Racket は，プログラミングの楽しさや面白さを感じやすいように工夫を凝らして開発された教育用の言語です．初心者に優しいユーザインタフェース，ゲームに特化したティーチパック等を備えており，今回の「楽しくプログラミングを学ぼう」という目的にぴったりな言語です．必要なものは，Windows または Mac OS X がインストールされているパソコンとサンプルプログラム（https://researchmap.jp/mutrf6bh8-2014482/#_2014482 からサンプルコードをダウンロードして解凍してください），そしてプログラミングへの興味だけです．プログラミング経験は，まったくなくても大丈夫です．本稿は，読みながら手を動かしていただければ，プログラミングを体験しつつ，同時に必要な知識を学べるようになっています．もちろん「プログラミング経験はあるけれども，Racket 言語は初めて」という方にも，お役に立つことでしょう．

図 -1　情報犬ビットくん

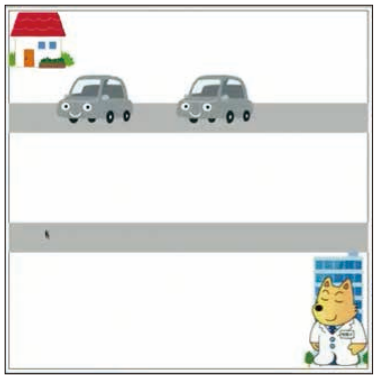

図 -2
未完成のゲーム画面

「ビットくんを，おうちに無事に帰らせる」というゲームを完成させよう

本稿では，「情報犬ビットくん☆1（図 -1）を勤務先の国立情報学研究所から動かし，車が走っている 2 本の道路を車にぶつからないよう横切らせ，おうちに帰らせる」というゲームを完成させます．ここでは「ビットくん帰宅ゲーム」と呼ぶことにしましょう．最初の状態では，ビットくんは左右には動けますが，上下には動くことができません．これでは，右下の情報研から左上のおうちまで動かすことはできません．また，止まっているときも歩いているときも，くつろいでいるはずのときも，ビットくんは同じ姿で，大変不自然です．さらに，2 本の道路のうち 1 本は，車が走っていません（図 -2）．ゲームを完成させるためには，

- ビットくんが立ち止まっているとき，歩いているとき，それぞれの動きに応じた姿にする

☆1「ビットくん」は国立情報学研究所（情報研）のマスコットキャラクタで，「情報研の情報犬」です．

図 -3　Racket の Web ページ

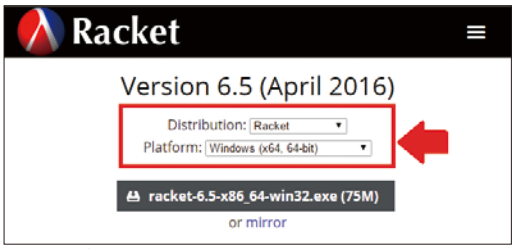

図 -4　ダウンロードのページ

- 2 本の道路に両方とも車を走らせる
- ビットくんが上下にも動けるようにする
- ビット君が車にぶつかったらゲームオーバーに，家に帰れたらゲームクリアにする

を行っていく必要があります．1 つ 1 つ，形にして（実装して）いきましょう．今回の前編では，上の 2 つを実装していきます．

Racket の準備

インストールと起動

それでは，「早くプログラミングを始めたい!」という気持ちをなだめながら，Racket をインストールしましょう．Web ブラウザで Racket の Web ページ https://racket-lang.org/ にアクセスし，右上の「ダウンロード」をクリックします（図 -3 の赤い矢印の部分）．すると，ダウンロードのページに移行します（図 -4）．インストールするものの種類（Distribution）は「Racket」と「Minimal Racket」の 2 つの選択肢のうち「Racket」を，環境（Platform）としてご自分のパソコン環境を選び，すぐ下に黒地に白文字で表示されるダウンロードボタンをクリックしてください．ダウンロードが終わったら，Windows の場合は，ダウンロードされたファイルを開き，指示に従ってインストールしていきます（図 -5）．Mac OS X では，ダウンロードしたディスクイメージを開き，表示に従って，アプリケーションに「ドラッグ・アンド・ドロップ」します．インストールの途中で「これを選択しておかなくてはならない」というものはありません．右下の「Next」ボタンをクリックしつづけると，「おまかせ」でインストール準備が行われます．最後に「Install」ボタンが出現したらクリックすると，インストールが始まります．インストール終了までに必要な時間は，長くても 5 分程度です．

図 -5　インストール中の画面（Windows）

図 -6　DrRacket のアイコン

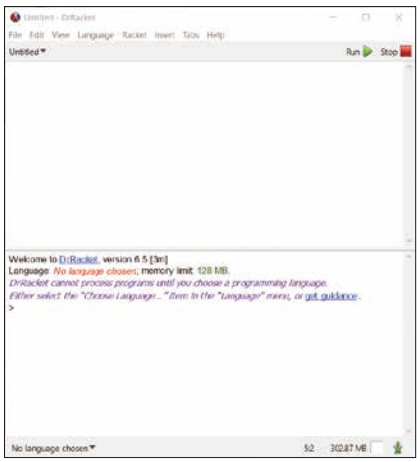
図 -7
Racket の画面

インストールが終了したら，それぞれの環境に応じて，「DrRacket」という名称のアイコンをクリックし，Racket を起動します．インストールした場所に「Racket（バージョン名）」というフォルダがあります．このフォルダを開くと，図 -6 のアイコン「DrRacket」が現れます．起動して，図 -7 のような画面が表示されたら，インストールは無事終了していることになります．もしも，素っ気ない真っ黒の画面が表示されたら，「DrRacket」ではなく「Racket」が起動されています．落ち着いて終了させ，改めて「DrRacket」を起動してください．

未完成のプログラムのダウンロード・環境設定・実行と終了を行ってみよう

それでは早速，未完成の「ビットくん帰宅ゲーム」のプログラムをダウンロードし，Racket 言語でプロ

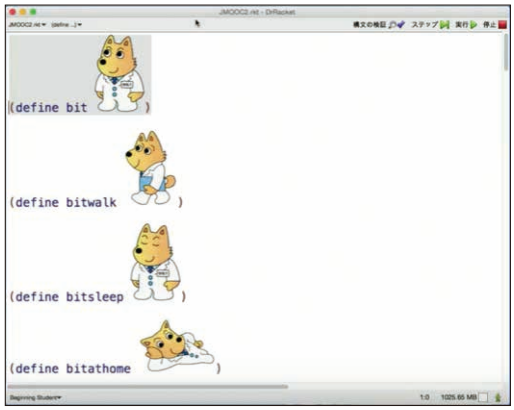

図 -8　サンプルプログラムを開いた状態


図 -10　サンプルプログラムの実行後の画面

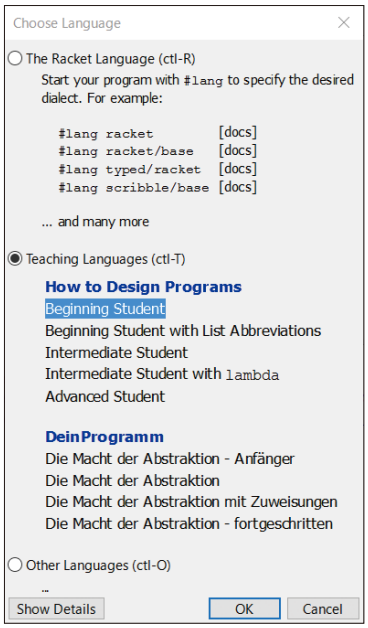

図 -9　言語選択のウィンドウ

グラミングして完成させるための準備を行います．まず，Web ページ（https://researchmap.jp/mutrf6bh8-2014482/#_2014482）から，使用するサンプルプログラムをダウンロードしてください．ダウンロードが終了したら，DrRacket の「ファイル (File)」→「開く ... (Open...)」で，ダウンロードしたファイルを開きます．上半分に図 -8 のような表示が現れたら，成功です．このようにプログラム表示の中に画像が出現するプログラミング言語やプログラミング環境は，実はとても珍しい存在なのです．これは，初めての人も画像の表示などで苦労せずに，面白いところをプログラミングできるようにという Racket の工夫の 1 つです．すぐに実行させてみたいところですが，その前に，必要な作業がいくつかあります．まず，プログラミング言語の種類を選択します．設定は，DrRacket の画面の左下，現在設定されている言語が表示されている部分（ダウンロードした直後は「No language chosen」となっている）の右側の「▼」をクリックし，選択メニューから「言語の選択 (Choose Language...)」をクリックすることで行います．すると言語を選択するためのウィンドウが現れるので，真ん中あたり，「学習用の言語 (Teaching Langugages)」の中の「Beginning Student」を選択し（図 -9），右下の「OK」をクリックします．次に，DrRacket 画面の上のメニューバーから「言語 (Language)」→「ティーチパックの追加 ... (Add Teachpack...)」で，必要な素材を追加するウィンドウを開きます．左・中央・右と 3 つの列があり，素材のリストが表示されています．真ん中から「universe.rkt」をダブルクリックして追加します．もう 1 つ追加するものがあるので，もう一度「言語 (Language)」→「ティーチパックの追加… (Add Teachpack...)」で同じウィンドウを開き，真ん中の列から「2htdp/image.rkt」をダブルクリックして追加します．これで準備は完了です．

では，プログラムの実行と終了を行ってみましょう．実行するには右上，右から 2 番目の「実行」をクリックします．少し待つと，別のウィンドウが開かれます（図 -10）．ウィンドウの中では，自動車が走っていて，プログラムが実行されていることが分かります．実行したプログラムを終了させるときは，開いたウィンドウの上部のバーにある終了ボタン（Mac OS X の場合はウィンドウ左上の赤い「×」ボタン）を押します．プログラムの実行と終了ができるようになれば，プログラミングで遊んだり試行錯誤して，自由自在に楽しめます．

プログラミングに踏み込んでみよう

それでは，「DrRacket」でプログラミングを行う部分に，何が表示されているのか見てみましょう．図 -11 のように，上と下，2 つのエリアに分かれており，それぞれに何かが表示されています．これらは，何なのでしょうか？　上のエリアは，「定義部分」と呼ばれます．プログラムの内容は，すべてこちらの「定義部分」に書いていきます．下のエリアは，「対話環境」と呼ばれます．本稿では，「エラーがあったら表示されるところ」と覚えておきましょう．試しに，上のエリアのプログラム

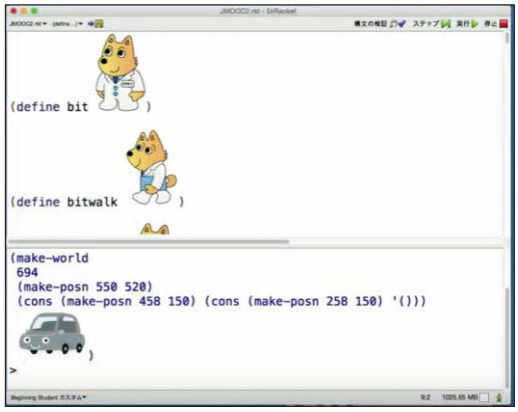

図 -11　Racket のエディタ画面

のどこかに，不要な閉じカッコ「)」を追加してみましょう．そして実行してみると，下のエリアに「read:unexpected ‘)’」というメッセージが赤く表示され，実行は中止されます．このように，下のエリアはエラーが起きたときだけ注目しましょう．

書いたプログラムを保存するには，ウィンドウの左上あたりのフロッピーの絵のアイコンをクリックするか，「DrRacket」の上のバーの左側，「ファイル」→「定義の保存」をクリックします．これで，DrRacket のエディタの基本的な使い方は，マスターできたことになります．

プログラム中の定数・関数の定義

では，プログラムを上から下まで，ざっと眺めてみましょう．全体は，たくさんのブロックに分かれています．ブロックのそれぞれは定数もしくは関数の定義です．定数は，

```
(define 定数名
        ……
)
```

という形です．ビット君の画像の定義などがこれにあたります．関数は

```
(define （関数名　引数……）
     ……
)
```

という形です．define のすぐあとの「(」の直後が，関数の名前，それに続くものがその関数が受け取るもの(引数)になります．関数は受け取る引数の値によって，返す値が変わります．定数・関数の具体的な例はこのあと見ていきましょう．

プログラムを少し変更してみよう

では，いよいよ，プログラミングです．もう一度このプログラムを実行し，「←」「→」キーを押してみてください．ビットくんが立った姿のまま，左右に動きます(図 -12)．これでは不自然です．それに，走ってくる車のスピードが遅すぎて，まったくスリルがありません．

まず，「歩いているときは，歩いている画像に」という変更を行ってみましょう．ビットくんが歩いている画像は，すでにプログラムの中にあります．上から 2 つ目の関数 (define bitwalk［歩くビットくんの画像］) は，「この画像に bitwalk という名前をつけますよ」という宣言です．プログラムのこの後の部分に bitwalk という単語が出てきたら，Racket は「この画像のことだね」と考えて，「表示する」などの処理を行ってくれます．では，ビットくんが歩いているときの画像は，どこで決められているのでしょうか？　ファイルの真ん中あたりの部分 (define (redraw world) …) というところに，redraw という関数があります．さまざまな場面で表示される画像は，この redraw 関数の中で決められています．redraw 関数の最後から数えて 3 行目に，歩いているときのビットくんの画像を決める部分があり，

```
(bit-image bit world)
```

と書かれています．このうち 2 つ目の bit，つまり立ち姿のビットくんの画像を，歩くビットくんの画像 bitwalk に変更して

```
(bit-image bitwalk world)
```

とすれば，歩いているときのビットくんは歩く姿になるはずです．この変更を行い，プログラムを保存してから実行して「←」「→」キーでビットくんを動かしてみると，左右に動かされているときには，歩いている画像になります (図 -13)．

ここで，もう 1 つ，簡単な変更を加えてみましょう．今の状態では，車のスピードが遅すぎて，ゲームとして，どうも楽しくなさそうです．スピードを変えるには，どうすればよいのでしょうか？


図 -12　正面の画像のまま横に歩くビットくん


図 -13　横向きの画像で歩くビットくん

プログラムの真ん中より少し末尾寄りに，move-on-tick という関数があります．これは，時間による動きを決める関数です．move-on-tick 関数の末尾から 2 行目に

```
(world-carlist (carmove world -1 0))
```

とあります．この -1 を -5 にし，保存して実行してみると，車はさきほどの 5 倍速になります．-20 にしてみると，クリアできなさそうな速さになります．車が遅すぎて簡単にクリアできるのも，速すぎてクリアが難しすぎるのも，ゲームとしては面白くありません．-5 くらいにしておきましょう．

ここでは，プログラムにちょっとした変更を加えてみました．実行してみると，表示されるものや動きが大きく変わります．このように，プログラムを少しずつ変更して行くことで，自分が思い描いている，自分の創りたいゲームに近づけていくことができます．

ゲームの「世界観」を 1 つのかたまりに〜「構造体」を使おう

ゲームの魅力を語る言葉の 1 つに「世界観」があります．この「ビットくん帰宅ゲーム」の世界観は，きわめて単純なものです．パラレルワールドが存在するわけではありません．ゲーム全体の世界の情報が，1 つにまとめられています．プログラミングでは，複数のデータを 1 つの「かたまり」として扱う「構造体」が，しばしば使われます．「ビットくん帰宅ゲーム」の世界の情報は，ただ 1 つの構造体でできています．プログラムの中央あたりに，

```
(define-struct world
  (walksec bitplace carlist carimg))
```

という部分があります[☆2]．「struct」は構造体という意味です．この宣言で新しい world という構造体を定義しています．その world 構造体は，ゲームに必要な情報を (walksec bitplace carlistcarimg) とひとかたまりにして持たせているのです．必要な情報の 1 つ目である walksec は，「ビットくんが歩いてから，どれだけ時間が経過しているか」です．ビットくんを「←」「→」で動かすと歩いている画像になり，しばらく動かさずになると立っている画像に，そのままさらに時間が経つと目をつぶった画像になります．「歩いてからどれだけ時間が経過したか」で，表示する絵を変えているわけです．2 つ目の bitplace は，ビットくんの位置です．3 つ目の carlist は，複数の車の座標のリストです．4 つ目の carimg は車の画像のデータです．これらゲームの世界を定義する構造体 world の中身を，キーボードの入力・時間の経過に従って書き換えることで，ゲーム全体が成り立っています．

では，最初に作られる「world」は，どのようなものでしょうか？　プログラムの末尾に big-bang という関数を呼び出している部分があり，ここで最初の「world」が作られています．

```
(big-bang
 (make-world 100 START-POSN
             (list (make-posn 500 150)
                   (make-posn 300 150)
                   ) cargray)
 (以下略))
```

Racket では，make-○○（○○ には構造体の名前が入る）とすると，構造体を作ることができます．さきほどの world 構造体の宣言と照らしあわせてみるために，抜き出してみましょう．

```
(make-world
   100 START-POSN (list(略)) cargray)
```

world 構造体の宣言は，

```
(define-struct world
  (walksec bitplace carlist carimg))
```

となっていましたね．1 つ目の 100 は，world 構造体の walksec に対応しています．つまり最初の状態では，ビットくんが歩き始めてからの時間は 100（1 秒間に 28 が対応していますので，100 はおよそ 4 秒です）ということです．2 つ目の START-POSN は world 構造体の bitplace に対応しています．これが，ビットくんの最初の位置です．プログラムの最初の方を見てみると，(make-posn 550 520) とあり (x, y)=(550, 520) となる位置であることが分かります．ちなみに Racket の座標系は左上が (0, 0) で，x 座標は右に行くほど大きくなり，y 座標は下に行くほど大きくなります．world 構造体の carlist にあたる 3 つ目の (list(略)) は，略さず書くと，

```
(list (make-posn 500 150)
      (make-posn 300 150))
```

☆2 実際のサンプルプログラムでは 1 行になっています．

です．同じような複数の何かを1つにまとめるとき，(list ....) というものを使います．上のリストには2つの車の座標が入っているため，最初に2台の車のある世界が作られるわけです．2台の車はそれぞれ，(make-posn 500 150) と (make-posn300 150) の座標，つまり (500, 150) と (300, 150) の位置に置かれます．4つ目の cargray は，world 構造体の carimg に対応しています．これは，車の画像です．このように，構造体を1つ定義し，最初の世界の状態（初期値）を与え，状態を書き換えていくことによって，ゲームが作り上げられています．

では，最初の世界を書き換えてみましょう．道路が2本あり，うち1本には車が2台走っています．車が走っていない方の道路にも，車を2台走らせてみます．このためには，今見たばかりの (big-bang……) 部分の内部を書き換えます．車の情報にあたる3つ目の list の中に，もう1本の道路を走る車を定義すればよいのです．

```
(big-bang (make-world 100 START-POSN
                      (list (make-posn 500 150)
                            (make-posn 300 150)
                            (make-posn 100 350)
                            (make-posn 400 350)
                            ) cargray)
                      (以下略))
```

と，2台分の車の最初の座標を追加し，プログラムを保存して実行してみると，車の数が4台になります（図 -14）．

このように，最初の状態（初期値）を変えると，ゲームの内容は変わります．また，ゲーム開始後の変化を違うものにすることによって，ゲームの内容を変えることもできます．先ほど行った車の速度の変更は後者にあたります．

それでは，構造体から1つ1つの要素を取り出すには，どうすればよいのでしょうか？　プログラムの中央より末尾寄りに，「move-on-tick」という関数があります．この関数の中は

```
(define (move-on-tick world)
  (make-world
    (+ (world-walksec world) 1)
    (world-bitplace world)
    (以下略)))
```

と，world-○○ world（○○には構造体の要素の名前が入ります）の形が4つ出てきます．この書き方で，world 構造体から指定した名前の要素を取り出すことができます．たとえばビットくんの位置には，bitplace という名前がついています．(world-bitplace world) で，world 構造体の中から，ビットくんの位置を取り出すことができます．

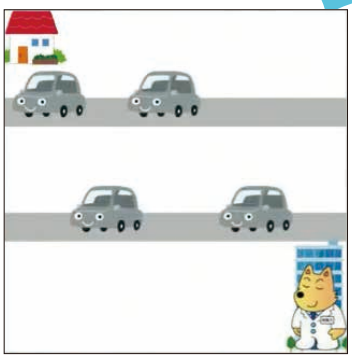
図 -14　車の数が増えたゲーム画面

このように構造体を使えば，複数のデータをひとかたまりで扱うことができます．本稿で使用した Racket の universe ティーチパックではゲームの世界の構造体を定義し，書き換えていくことで，ゲーム全体が成り立っています．

後編について

すんなりうまくできましたでしょうか．うまくいかなかったとしてもそれに対処した経験は後で役立つものです．失敗することを恐れずに，気軽にプログラミングを楽しんでいきましょう．本稿では，Racket プログラミングの始め方を解説し，プログラムのちょっとした変更で実行結果が変わることを体験して，ゲームの世界を作っている構造体というものを学んでいきました．次号の後編では

- ビットくんが上下にも動けるようにする
- ビット君が車にぶつかったらゲームオーバーに，家に帰れたらゲームクリアにする

の変更を通して，複雑な条件分岐と，新しい関数の書き方を学んでいきます．また次号でお会いしましょう．

（2016年5月9日受付）

対馬かなえ（正会員）■ k_tsushima@nii.ac.jp
国立情報学研究所 助教．関数型言語，型システム等に興味を持つ．現在，プログラミングを楽にすることを目的に，デバッグ手法について研究している．