平成 21 年 11 月 21 日
京都工芸繊維大学コンピュータ部

Lime 40

はじめに

こんにちは。部長の荒木です。これを読んでいただいているということは、今年も無事、Lime を発行することができたということでしょう。

さて、Lime も今回で第 40 号です。Lime は、部員の活動の一部を記したものです。日々の研究や開発の成果、興味を持った事に対する調べ物や、果ては個人的趣味全開なもの。是非、最後までお付き合いください。

平成 21 年 11 月 19 日
京都工芸繊維大学コンピュータ部部長 荒木 修

目 次

1 ストリームの動向を CLOS を交えて

情報工学課程 4 回生 湯浅 信吾

1.1 はじめに

本記事ではストリームの動向を CLOS を交えて話します。「おいおい。『ストリームの動向』って “stream” は『動向』という意味じゃないか！　あと『交える』は “cross” だろ！」と突っ込んでいただけたら幸いです。対象とする読者は “Common Lisp をちょっとだけ知ってる方”、“他の Lisp 方言を知っている方”、“Lisp は知らないけど妙に理解が早い方” ですが、それ以外の方にも読んでいただけたら嬉しいです。

1.2 ふつうのストリーム

1.2.1 ストリームとは

Common Lisp には**ストリーム**というものがあります。ストリームとは「データの読み込み元または書き込み先として用いられるオブジェクト」です。文字を扱うものは**文字ストリーム**、整数を扱うものは**バイナリストリーム**といいます。

文字ストリームから 1 文字読み込むには関数 read-char、1 文字書き込むには関数 write-char を使います。バイナリストリームから 1 バイト読み込むには関数 read-byte、1 バイト書き込むには関数 write-byte を使います。これを図 1.1 に示します。

他にもストリームのための関数、マクロは多数ありますが、そのほとんどが、ストリーム一般に使えるものか、文字ストリーム専用のものです。バイナリストリーム専用のものは上記 2 関数以外存在しません。

プログラマはストリームに対して関数を呼び出せばよく、ストリームがどのようにファイルなどとやり取りをしているかを意識する必要はありません。

1.2.2 ストリームの一生

ストリームは make-xxx-stream といった名前の関数やマクロ、関数 open などによって生成されます。生成されたストリームは常に**オープン**されています。ストリームを使い終えると、関数 close によってストリームを**クローズ**します。クローズされたストリームに対して入出力操作を行うこと

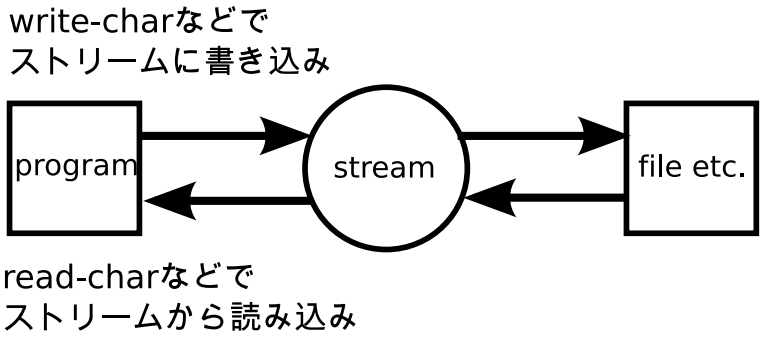

図 1.1: ストリーム

はできません。

1.2.3 ファイルストリーム

ファイルに対して読み書きを行うストリームを**ファイルストリーム**といいます。ファイルを開くときに扱うデータの種類を指定できるので、ファイルストリームは文字ストリームにもバイナリストリームにもなり得ます。テキストファイルを扱うときは文字ストリームとしてファイルストリームを作り、バイナリファイルを扱うときはバイナリストリームとしてファイルストリームを作ります[1]。

関数 open によりファイルストリームを生成できます。以下に使用例を示します。

```
;; Open text file
(setf c-stream (open "FILENAME"
                     :direction :input
                     :element-type 'character))
(print (read-char c-stream))  ;Read first character
(close c-stream)
```

開いたストリームは使い終わったらクローズする必要があります。非局所脱出が行われた際にもクローズするために、特殊形式 unwind-protect を使うのがいいでしょう。

```
;; Open binary file
(let ((b-stream (open "FILENAME"
                      :direction :input
                      :element-type '(unsigned-byte 8))))
  (unwind-protect
      (print (read-byte b-stream))  ;Read first byte
    (close b-stream)))  ;Close stream always
```

[1]Common Lisp の仕様の範囲内では、文字とバイナリの両方を同時に扱う手段は提供されていません。そのため、処理系ごとに独自の拡張が施されており、例えば、CLISP の場合は、ストリームを開いてからその種類を切り替えることが可能で、SBCL の場合は文字とバイナリを同時に扱える Bivalent Stream というものが利用できます。

unwind-protect と open、close といった一連の式を生成してくれるマクロ with-oepn-file を使用すると、もっと楽に書くことができます。

```
;;; Read binary file
(with-open-file (b-stream "FILENAME"
                 :direction :input
                 :element-type '(unsigned-byte 8))
  (print (read-byte b-stream)))  ;Read first byte
```

これは unwind-protect を使って書いた式とほぼ同等の式に展開されます。

1.2.4 文字列ストリーム

文字ストリームとは別に、**文字列ストリーム**というものがあります。これは、文字列に対して読み書きを行うストリームです。文字列ストリームは文字ストリームの一種です。これまでに出てきたストリームの関係を図 1.2 に示します。

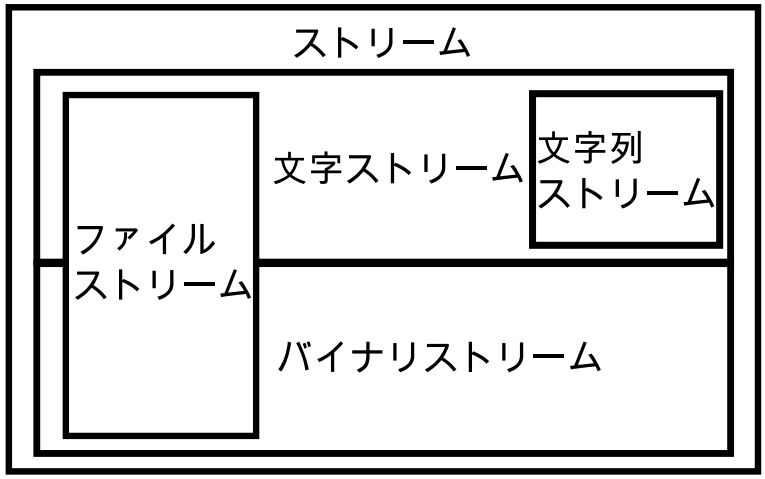

図 1.2: ストリームの関係

文字列ストリームは関数 make-string-input-stream によって生成できますが、with-oepn-file のようにストリームのクローズを自動的に生成してくれるマクロ with-input-from-string を使う方が便利です[2]。

文字列ストリームは文字ストリームを扱う関数をテストするとき便利です。例として、「テキストファイルを読み込み、そこから括弧 "()" を取り除いたものを標準出力に書き出すプログラム」を作る場合を考えます。

```
(defun print-kakko-igai (input)
```

[2] これらは文字列から文字を読み込むストリームを生成するものです。文字列に文字を書き込むストリームを生成するには make-string-output-stream、with-output-to-string を使用します。

```
(do ((c (read-char input nil :eof)
        (read-char input nil :eof)))
    ((eq c :eof) t)
  (unless (member c '( #\( #\)))
    (princ c))))
```

関数 print-kakko-igai はストリーム *input* を受け取り、*input* から 1 文字ずつ文字を読み込み、それが括弧以外であれば標準出力に書き出します。

あとは、ファイルストリームを開いて、print-kakko-igai を呼び出す部分を作れば完成ですが、できればこの関数単体でテストをしたいものです。このような場合に文字列ストリームが使えます。

```
(with-input-from-string (s "(cadaar '(((1 2) (3 4)) 5 6))")
  (print-kakko-igai s))
```

この式を評価すると「cadaar '1 2 3 4 5 6」が出力されます。マクロ with-input-from-string により、文字列ストリーム *s* が作られ、関数 print-kakko-igai はそこから 1 文字ずつ読み込むという訳です。

1.2.5 かわいそうなバイナリストリーム

バイナリファイルを読み込むプログラムを書くときも、関数単位での小さいテストをしたいものです。しかし、with-input-from-string のバイナリストリーム版は存在しません。

それに代わるものを自作しようにも、Common Lisp の仕様の範囲内では、そのようなものを作ることはできません。図 1.1 の “stream” より右側を操作することはできないんです。文字ストリームは色んな便利な機能が用意されているというのに、バイナリストリームにできることは read-byte と write-byte だけ。ああ、なんてかわいそうなバイナリストリーム。

1.3 Gray Stream

1.3.1 Gray Stream とは

Common Lisp はユーザが標準の I/O 関数を使用する独自のストリームを作る手段を提供していません。しかし、独自のストリームを作る手段は仕様に採択されなかったものの、STREAM-DEFINITION-BY-USER[1] という名前で提案されていました。

今ではこれは Gray Stream という名前で、多くの処理系で実装されています[3]。Gray Stream の基本的な考えは次のようなものです。

- I/O のための総称関数を定義
- 既存の I/O 関数はそれらの総称関数の呼び出しを行う

[3]私が調べた限りで、ACL、SBCL、CMUCL、CLISP、ECL、Clozure CL に実装されています。また、LispWorks にも “User Defined Streams” という名称で同様のものが実装されています。

- ユーザは独自のストリームクラスのためのメソッドを追加できる

I/O のための総称関数は stream-read-char、stream-write-char、stream-read-byte、stream-write-byte のように、既存の I/O 関数に “stream-” を付けた名前となっています。これらの関数はそれぞれ write-char、read-char、read-byte、write-byte から**呼び出され**ます。これを図示すると図 1.3 のようになります。

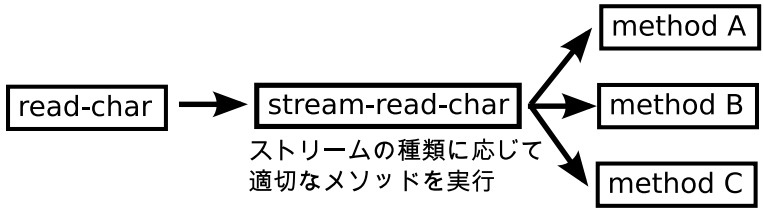

図 1.3: Gray Stream の仕組み

1.3.2 総称関数

話を一度 Gray Stream から総称関数に移します。**総称関数**とは与えられた引数の種類によって振る舞いを変える関数です。通常の関数は本体（関数の行う一連の処理）を 1 個だけ保持し、関数が呼び出されるとその本体を実行します。一方、総称関数は本体を複数保持します。その本体 1 つ 1 つを総称関数の**メソッド**と呼びます。以下に総称関数の定義の例を示します。

```
(defgeneric gf1 (arg))

(defmethod  gf1 ((arg number))
  (print "Number desu"))

(defmethod gf1 ((arg symbol))
  (print "Symbol desu"))
```

マクロ defgeneric は総称関数を作ります。上記の使い方の場合、これが呼ばれた時点では総称関数 gf1 はメソッドを持ちません。マクロ defmethod は総称関数にメソッドを追加します。メソッドは引数の属すクラスを指定することができ、総称関数を呼び出す際には**実行時に**引数の属すクラスが調べられ、適切なメソッドが呼び出されます。

```
(gf1 2345)  ;=> "Number desu"
(gf1 'sym)  ;=> "Symbol desu"
```

総称関数は通常の関数と同様に扱うことができ apply や funcall の引数として使うこともできます。

1.3.3 クラスの継承と総称関数

先ほど「クラス」という言葉を使ったことから分かるように、総称関数は Common Lisp のオブジェクト指向機能拡張である CLOS (Common Lisp Object System) の機能です。ここで、クラスを自分で作ってみましょう。

```
(defclass A () ())   ;A is superclass of B
(defclass B (A) ())  ;B is subclass of A

(defgeneric F (obj))
(defmethod F ((obj A))
  (print "A desu"))
(defmethod F ((obj B))
  (print "B desu"))

(defgeneric G (obj))
(defmethod G ((obj A))
  (print "A kana?"))
```

クラス A、B と総称関数 F、G を定義しました。F は A のためのメソッドと B のためのメソッドを保持していますが、G は A のためのメソッドしか保持していません。これらを呼び出してみます。

```
(setf a-inst (make-instance 'A))
(setf b-inst (make-instance 'B))
(F a-inst)  ;=>"A desu"
(F b-inst)  ;=>"B desu"
(G a-inst)  ;=>"A kana?"
(G b-inst)  ;=>"A kana?"
```

総称関数の呼び出しを行うと、実引数の属すクラスがマッチするメソッドがない場合、実引数の属すクラスのスーパクラスがマッチするメソッドが呼び出されます。そのため、総称関数 G に対して B のインスタンスを適用すると、A のためのメソッドが呼ばれます。CLOS でのメソッドの継承はこのように、総称関数により暗黙的に行われます。

ただし実際は、CLOS のクラスは多重継承を許しますし、総称関数は複数の引数を取ることが可能なので総称関数のディスパッチの仕組みはかなり複雑です。ここではその詳細は割愛させていただきます。

1.3.4 Gray Stream のクラス

話を Gray Stream に戻します。既存の I/O 関数を総称関数にするのではなく、新たに総称関数を定義し、それを呼び出すのは、既存の I/O 関数の多くがストリームをオプショナル引数として取るためです[4]。新たに定義する総称関数は引数としてストリームを必ず取り、その型に応じて適切なメソッドを起動できるようになっています。

[4]例えば、read-char の引数は省略可能で、その場合は標準入力から 1 文字読み込まれます

全ての Gray Stream の基本となるクラスとして、fundamental-stream というクラスが提供されます。これを継承して独自のストリームクラスを作ることができます。また、これ以外にもいくつかの基本的なストリームのクラスが提供されます。それらは全て fundamental-stream のサブクラスです。

1.3.5 バイナリストリームを拡張する

Gray Stream を使って、文字列ストリームのバイナリ版を作ってみます。まずは、新たなストリームのクラスを作ります。文字列ストリームのバイナリ版ということで「バイナリ配列ストリーム」とでも名付けます。

```
(defclass binary-array-input-stream
    (fundamental-binary-input-stream)
  ((array :initarg :array :type (array * (*)))
   (index :initarg :start :type fixnum)
   (end :initarg :end :type fixnum)))
```

binary-array-input-stream のスーパクラス fundamental-binary-input-stream は扱う型が unsigned-byte か signed-byte のサブクラスである入力ストリームのクラスです。スロット array の型は緩く設定していますが、これはリーダマクロ #(で作った配列を利用できるようにするためです。

次に、このクラスのメソッドを書きます。

```
(defmethod stream-read-byte ((stream binary-array-input-stream))
  (with-slots (index end array) stream
    (if (>= index end)
        :eof
        (prog1 (aref array index)
          (incf index)))))
```

これで、read-byte の引数がバイナリ配列ストリームであれば、このメソッドが呼ばれるようになります。内容は単純なもので、配列から 1 個ずつ要素を読み出し、終端まで行けば:eof を返す[5]というものです。

最後に、with-input-from-string に相当するマクロを書きます。

```
(defun make-binary-array-input-stream (array &optional (start 0) end)
  (make-instance 'binary-array-input-stream
                 :array array :start start :end (or end (length array))))

(defmacro with-input-from-binary-array ((var array) &body body)
  `(let ((,var (make-binary-array-input-stream ,array)))
     (multiple-value-prog1
         (unwind-protect
              (progn ,@body)
           (close ,var)))))
```

[5]Gray Stream ではストリームの終端に到達すると、:eof を返すという規則になっています。

これで完成です。使用例として「バイナリファイルから最初の 2 バイトを読み込み、それをビッグエンディアンとして扱った場合の数、リトルエンディアンとしてとして扱った場合の数を表示するプログラム」を考えます。

```
(defun byte-list->number (byte-list &optional little-endian-p)
  (reduce #'(lambda (high low) (+ (ash high 8) low))
          (if little-endian-p
              (reverse byte-list)
              byte-list)))

(defun read-byte-list (n stream)
  (let (byte-list)
    (dotimes (_ n (nreverse byte-list))
      (push (read-byte stream) byte-list))))

(defun big-and-little (input)
  (let ((byte-list (read-byte-list 2 input)))
    (format t "big endian: ~A~%" (byte-list->number byte-list))
    (format t "little endian: ~A~%" (byte-list->number byte-list t))))
```

関数 byte-list->number は整数のリストを受け取り、これをバイト列と解釈し、適切な整数を返します。関数 read-byte-list はストリームから複数バイト読み込み、それをリストの形で返します。関数 big-and-little はこれら 2 個の関数を使って、目的の動作を行います。

あとはファイルストリームを開いて big-and-little を呼び出す部分を作れば完成ですが、テキストファイルの時と同様に、big-and-little 単体でテストをしたい場合、with-input-from-binary-array が利用できます。

```
(with-input-from-binary-array (s #(1 44))
  (big-and-little s))
```

これを評価すると「big endian: 300」と「little endian: 11265」が表示され、正しく動作することが確認できます。

1.3.6 Gray Stream の問題点

ストリームに拡張性を与えてくれた Gray Stream ですが、いくつかの問題を抱えています [2]。

- ストリームの入出力の方向をクラスで区別する必要がある[6]
- 文字ストリームとバイナリストリームを区別するのは面倒
- CLOS のディスパッチ処理のコストがかかる
- メソッドの継承が複雑になる

[6]先ほど作ったクラスは fundamental-input-stream を継承しました。出力用ストリームを作るには fundamental-output-stream を継承する必要があります。

- コードが重複する

CLOS を使うことによって拡張性を得ましたが、CLOS によって新たな問題が発生してしまいました。

1.4 Simple Stream

1.4.1 Simple Stream とは

Gray Stream の問題点を解決した拡張可能なストリームとして Simple Stream[2] というものがあります。これは Allegro Common Lisp(ACL) の Version 6.0 で追加されたものです。Simple Stream は ACL の他に、SBCL でも使用できますが、まだ部分的な実装しか行っていない α 版のようです。

Simple Stream はもともと ACL の Common Graphics というウィンドウシステムで使われていたものを、一般のストリームとして使えるように拡張したものです。[1] でも「ポータブルなウィンドウシステムの開発のために、ユーザがストリームを定義する手段が必要」と書いてます。ストリームはウィンドウシステムとともに発展するのかもしれません。

1.4.2 Simple Stream の特徴

Simple Stream の特徴として次のようなものが挙げられます。

- 入力/出力はフラグによって区別する
- element-type を区別しない
- 外部入出力装置と疑似デバイスを区別する
- バッファに対する細かい処理が可能

Simple Stream は element-type を区別せず、すべてオクテット (8-bit byte) であるかのように[7]扱います。これにより、文字とバイナリを同時に使うことができます。また、Simple Stream は速度を重視しており、低レベルの開発が可能となっています。これが Simple Stream 最大の特徴です。

1.4.3 Simple Stream と Gray Stream の共存

標準の Common Lisp の機能しか使ってないのであれば、ストリームを全て Simple Stream にすればいいのですが、Gray Stream によりストリームの拡張を行っている場合、Gray Stream と Simple Stream を共存させる必要があります。

ACL の read-char などの関数は、与えられたストリームが Gray Stream であれば stream-read-char などの総称関数を呼び出し、Simple Stream であれば Simple Stream 専用の処理を行います。

[7] もちろん、実際には多バイト文字なども正しく扱えます。

また、stream-read-char などの総称関数の呼び出しをしているコードを Simple Stream を使ったコードに直したい場合は、Gray Stream をシノニムストリームに置き換えてやると、stream-read-char から、read-char が呼び出されるので、変更箇所を少なくすることができますが、この詳細の説明は割愛させていただきます。

1.5 おわりに

1.5.1 まとめ

ストリームに拡張性がないと困るということで Gray Stream が生まれ、Gray Stream には色々問題があるということで Simple Stream が生まれました。現在のところ、多くの処理系が Gray Stream を実装しており、Simple Stream を実装しているのは ACL だけです。Gray Stream が提供してくれるのは拡張性ですが、Simple Stream が提供してくれるのは主に速さのようです。拡張性を欲しがる人は多いでしょうが、I/O の速さとなると、欲しがる人は幾分か減るのではないでしょうか。よって、Simple Stream が現在の Gray Stream のように数多くの処理系で実装されることは当分はないと私は予想します。ただし、この予想には一切責任を持ちませんのでご了承を (笑)

1.5.2 蛇足

1〜3 回生までは Lime には自分が作ったもののことを書いてたんですが、4 回生になって、初めて人が作ったもの (?) について書くことになりました。本当は自分で作ったもののことを書きたかった。つい最近、Java のアセンブラと逆アセンブラを Common Lisp で書いたんですが、Java などという S 式も使わない汚らわしい言語のことなんてとても書けない …… のではなく、力技の汚いプログラムを書いただけなので、特に記事にするところがないということで見送りました。「マクロを使って Java VM の命令定義からアセンブラと逆アセンブラ両方のコードを生成」とかやったら少しは面白かったんだろうなと完成後に気づきました。大学に入ってから Lisp ばかり触ってきましたが、まだまだ Lisp をうまく使いこなせていないようです。Lisp の 51 年という長い歴史を考えると、私はまだ Lisp という長い、長い坂道を登り始めたばかりなのかもしれません。

参考文献

[1] Kent Pitman
「FAILED Issue STREAM-DEFINITION-BY-USER (“Gray Streams”)」
<http://www.nhplace.com/kent/CL/Issues/stream-definition-by-user.html>

[2] Franz Inc
「Streams in Allegro CL」
<http://www.franz.com/support/documentation/6.2/doc/streams.htm>

2 ウェブカメラ+OpenCV で何か...

情報工学課程 3 回生 村上 明男

2.1 はじめに

この間ウェブカメラを買いました。そこで、これを使って何かできないかと思い、画像処理ライブラリの OpenCV を使ってみることにしました。

... で、

ある日、

> oa [1]:よし、ウェブカメラと OpenCV で音ゲー[2]を作ろうじゃないか。
> ika [3]:...。

ということで、音ゲーをつくることに。

2.2 OpenCV とは

OpenCV(Open Source Computer Vision Library) は代表的な画像処理をライブラリ化したもので、Intel 社によって開発され、現在は Willow Garage 社によって開発が進められています。

2.3 開発準備

2.3.1 構想

ゲーム的には、音楽に合わせて、的が出てくるから、その的にタイミングよく座標 (なんらかの入力で動く) を合わせる。方法として、ウェブカメラで画像をとって、画像の特定の色を OpenCV で画像処理して座標として抽出、それをゲームロジックに渡す。

2.3.2 開発分担

開発だが、まずは、音ゲーを作るために開発の分担をしなければならない。

[1]「荒木 修」という人間の名前。
[2]音楽ゲームのこと。音楽に合わせて、プレイヤーがアクションをとることで進行するゲーム。「コナミ」の「ビートマニア」などが有名
[3]「村上 明男」という人間の名前。

oa:よし、おいらはゲームロジックを作るから、ika さんは、カメラからの入力部分を作るんだ!!
ika:...。

ズバリ、主要な要素は 2 つ。

1. ウェブカメラと OpenCV を使って、特定の色から、入力座標を生成する部分
2. 入力座標を受けとって、実際のゲーム処理を行う部分

前者を村上、後者を荒木が担当することに。

2.4 特定の色を抽出

2.4.1 HSV 表色系の利用

特定の色を抽出したいので、その画素が特定の色であるかどうかを判定するための閾値を設定する。ここで、閾値の設定を少しでも楽にしたいので、表色系を RGB から HSV に変換する。RGB がそれぞれ赤色の輝度、緑色の輝度、青色の輝度を数値で表す表色系であるのに対し、HSV はある色を色相 (Hue)、彩度 (Saturation)、明度 (Value) の 3 つの数値で表す表色系である。
カメラから取り込まれた画像は BGR で格納されているのでここでは BGR から HSV へ変換する。

```
cvCvtColor( frameImage, hsvImage, CV_BGR2HSV );
```

cvCvtColor 関数によって、BGR 形式の画像 frameImage は HSV 形式へ変換されて、画像 hsvImage が出力される。

2.4.2 画素値の変更

HSV 表色系へ変換した後、画像から 1 画素を取り出し、さらに H、S、V それぞれの値を取り出す。次に、それぞれの値が閾値の範囲内であるかどうかを判断する。H、S、V のすべての値で閾値の範囲内 (特定の色である) なら白、範囲外 (特定の色ではない) なら黒と表示される画像を生成するのがここでの目的。(実質 2 値化)

```
color = cvGet2D( hsvImage, y, x );
cvSetReal2D( extractionImage, y, x, 255 );    //    閾値範囲内
cvSetReal2D( extractionImage, y, x, 0 );    //    閾値範囲外
```

cvGet2D 関数で画像 hsvImage の 1 画素から画素値を取り出し、cvGetReal2D 関数で画素値を変更する。cvGetReal2D 関数は閾値によって使い分ける。

2.4.3 メディアンフィルタをかける

メディアンフィルタとは、対象となる 1 画素 (中心画素) と周囲の 8 画素の合計 9 画素の画素値を昇順あるいは降順に並べ、中央の値を新たな画素値とすることで小さなノイズを除去できるフィルタのことで、ごま状のノイズを除去したい時に効果を発揮する。

```
cvSmooth(extractionImage, MedianImage,
          CV_MEDIAN, 5, 0, 0, 0);
```

これで画像 extractionImage をメディアンフィルタにかけ、画像 MedianImage に出力できる。

2.4.4 実行の流れと実行例

実行の流れは以下の通り。

1. キャプチャ用画像、格納用画像のためのメモリを確保
2. ウィンドウの作成
3. 画面を 1 フレームキャプチャ
4. HSV 形式への変換
5. 画素値の変更 (閾値で分岐)
6. メディアンフィルタの適用
7. ウィンドウに画像を表示
8. 3〜7 の繰り返しの終了後、メモリを解放
9. ウィンドウの破棄

また、以下のようにキーボードで指定したキーを入力することで特定の動作を行わせることもできる。

```
int key;

//'q' キーが入力されたらループを抜ける
key = cvWaitKey(10);
if(key == 'q'){
    break;
}else if(key == 'c'){
    //    'c' キーが押されたら円を表示・非表示切り替え
    if(flag == 0){
        flag = 1;
    }
    else{
```

```
            flag = 0;
        }
    }else if(key == 'p') {
        //    'p' キーが押されたら画像を保存
        cvSaveImage( "D:/openCVpict/frame.jpg",
                     frameImage );
        cvSaveImage( "D:/openCVpict/extraction.jpg",
                     extractionImage );
        cvSaveImage( "D:/openCVpict/Median.jpg",
                     MedianImage );
    }
```

以上のようにすればプログラムの終了、円の表示・非表示 (円については後述)、現在の画像の保存をキーボード入力で操作できる。
元の画像、画素値変更後の画像、メディアンフィルタ適用後の画像をそれぞれ以下に示す。

図 2.1: 元の画像

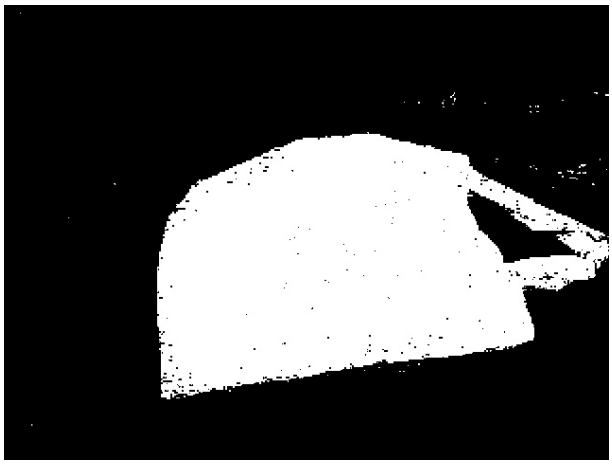

図 2.2: 画素値変更後

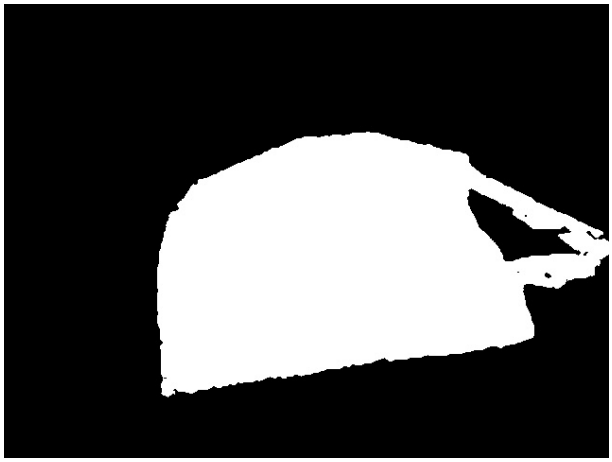

図 2.3: メディアンフィルタ適用後

元の画像から緑色を抽出したところ、緑色のバッグの部分が抽出された。そしてメディアンフィルタが細かいノイズを消してくれていることも確認できた。

2.4.5 重心に円を表示させる

せっかくなので、抽出した緑色の部分を抽出したついでに、抽出部分の重心を求めてみる。

```
cvMoments( MedianImage, &moment, 0);
m_00 = cvGetSpatialMoment( &moment, 0, 0 );
m_10 = cvGetSpatialMoment( &moment, 1, 0 );
```

```
m_01 = cvGetSpatialMoment( &moment, 0, 1 );
GravityX = (int)(m_10 / m_00);
GravityY = (int)(m_01 / m_00);
```

cvMoments 関数で 3 次までのモーメントを計算して CvMoments 構造体に格納する。その後、重心の計算に用いるモーメントを cvGetSpatialMoment 関数で取り出し、最後に重心を求める。

次に、円を描きたい画像と円の中心座標 (ここでは重心)、円の半径などを指定して cvCircle 関数を使う。その後に、cvShowImage 関数を使うことで円が描かれた状態の画像 frameImage を表示させることができる。

```
cvCircle( frameImage, cvPoint( GravityX, GravityY ),
          CIRCLE_RADIUS, CV_RGB( 0, 255, 0 ), LINE_THICKNESS,
          LINE_TYPE, 0 );

cvShowImage(windowNameCapture, frameImage);
```

これで緑色の円が抽出部分の重心の位置に表示される。

2.5 ゲームロジック

今回制作する音ゲーでは、さきほど述べた特定の色の抽出はバックエンドで行う。そして、抽出部分の重心を入力座標としてゲームロジックに渡す。

実際にゲームロジックが、座標の取得でやり取りする関数は、初期化を行う関数、毎フレームごとに座標を取得する関数、終了してメモリを解放する関数の 3 つで済む。

```
void captureStart(); //main 関数の最初に呼び出す
void callCoordinate(int* x, int* y); //ゲームループ毎に呼ぶ
void releaseMemory(); //main 関数の最後に呼び出す
```

さて、次にゲームロジックだが、担当ではないので、バトンタッチ。

oa: 説明が面倒だ。実際にゲームをプレイする部分のアルゴリズムはこんな感じだ。

1. 音楽を再生し始める
2. プレイヤーの座標を更新
3. 出現時間となった的を出現させる
4. プレイヤーの座標と的の当たり判定 (当たっていた場合、的は消滅)
5. 的の動作処理 (出現時間を終えた的は消す)
6. 的の描画
7. プレイヤーの座標の描画

8. 音楽再生が終了すれば、ゲーム終了

9. 2〜8 のループ

oa: ソースコードはあまりに汚すぎてみせられないの。見ても、楽しくないし。
oa: 実行してみたら、こんなかんじ。色々追加したあとのやつだけどね (古いのはなくした orz)

図 2.4: ゲームのプレイ画面

oa: とりあえず、この時点 (11 月 14 日) では、まともにウェブカメラからの操作がうまいこといかなかった。ふむ、難しい。

2.6 おわりに

座標の取り方についてはもう少し工夫をすればうまくいく... と思い、試行錯誤しています。

- 重心がブレる→過去の重心座標と組み合わせればなめらかになるかも
- 閾値が光の加減で変わる→その都度修正
- 背景に影響される→背景の動的更新

この記事はこれで終わりですが、プログラムの修正は続きます。

ika: そして人生も続くんですねぇ...
oa: まだまだ終わらんよ。

THE END?

参考文献

[1] 奈良先端科学技術大学院大学 OpenCV プログラミングブック制作チーム
『OpenCV プログラミングブック 第 2 版』 pp.14-17.58-73.241-242.271.348-350. 毎日コミュニケーションズ

[2] 「OpenCV-1.1pre リファレンス マニュアル（日本語訳）」
<http://opencv.jp/opencv-1.1.0/document/>

3 迷路を自動生成してみた

情報工学課程 3 回生 出原 真人

3.1 始めに

毎年、この時期になると学園祭に展示する作品の製作に追われています。去年は 3D のオセロゲームと迷路ゲームを作りましたが、迷路ゲームは固定マップだったのが心残りでした。今年も去年に引き続きミニゲーム集を製作中なので、去年の雪辱戦とばかりにマップが自動生成される迷路ゲームをつくることにしました。

3.2 アルゴリズム

3.2.1 アルゴリズムの種類

迷路生成のアルゴリズムには以下のようなものがあります。

- 棒倒し法
- 道のばし法
- 壁のばし法

この他にもいくつかあるみたいですが、今回は出来上がる迷路のバリエーションに比較的幅のある壁のばし法を用いることにします。

3.2.2 壁のばし法

プログラムを載せる前に、壁のばし法のアルゴリズムを簡単に解説しておこうと思います。

- 最初に、図 3.1(a) のように外周を壁で囲うように初期化します。
- 次に、ランダムに選んだ点から壁を伸ばします。選んだ点が既に壁だった場合は、次進むと別の壁に繋がってしまうという点まで、道だった場合は別の壁に繋がる点まで伸ばします。例えば、図 3.1(b) において真ん中の方の点から始まっていれば右の壁に繋がって終了、右の壁が開始点ならば左端の点で上に進もうとして、行き止まりで終了となります。

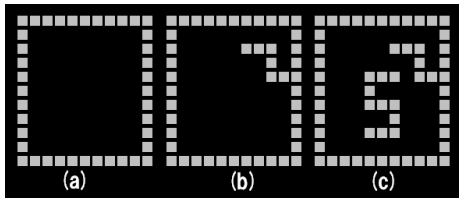

図 3.1: 迷路生成の様子

- 開始点が道の場合、別の壁を目指している最中に自分自身にぶち当たることがあります (図 3.1(c) なら左下の点から次は上に進もうとしている)。こういう場合は伸ばしている最中だった壁を取り壊し、開始点を選び直します。
- 作業を繰り返し、おけなくなったら迷路完成です。

3.3 ソースコード

ここからは実際に作成したソースコードを解説していきます。

3.3.1 とりあえず準備

```
#define WALL 0
#define NONE 1
#define TEMP 2
typedef struct _map{
    char **map;
    int width , height;
}MAP;
typedef struct _vec {
    char x;
    char y;
}VEC;

VEC vec[4]={{0,-1},{1,0},{0,1},{-1,0}};
MAP stg;
```

色々と必要になってくるものを定義や宣言しておきます。VEC 構造体は進めるかどうかのチェックに、MAP 構造体はマップデータを格納するのに使います。マップデータは 2 次元配列で確保します。実際の迷路の構造とマップデータは図 3.2 のように対応付けることにします。

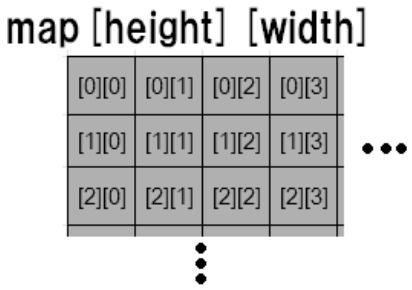

図 3.2: マップデータ

3.3.2 初期化と基本ループ

```
void make_maze(MAP *map){
    int y,x;
    srand((unsigned int)time(NULL));
    while( check_end(map) ){
        y = rand() % map->height;
        x = rand() % map->width;
        if((x%2 || y%2)) continue;
        switch ( map->map[y][x] ){
        case NONE:/*開始点が道の場合*/
            make_wall_B(map,y,x);
            break;
        case WALL:/*開始点が壁の場合*/
            make_wall_A(map,y,x);
            break;
        }
    }
}

void load_map(MAP *map){
    FILE *fp;
    int y,x;
    /*配列の確保*/
    map->map = (char**)calloc(map->height,sizeof(char*));
    for(y=0 ;y < map->height ; ++y){
        map->map[y] = (char*)calloc(map->width,sizeof(char));
    }
    if(map->map == NULL){
        printf("callocError\n");
        return;
    }
    /*外周を壁にする*/
    for(y=1;y < map->height-1 ;++y){
        for(x=1; x< map->width-1 ; ++x){
```

```
                map->map[y][x] = NONE ;
            }
        }
        make_maze(map);
    }
```

load_map は配列の確保と初期化を行う関数です。あらかじめ map->width,map->height を設定してからこの関数を呼び出す必要があります。マップの大きさは縦横それぞれ奇数にしておきましょう。偶数でも可能ですが一番端が汚くなります。

make_maze 内で終了判定の check_end が偽を返すまでループを回します。ループ内では開始点を決定し、make_wall 系統の関数を呼び出して実際に壁を作ります。

尚、最終的には縦横座標が両方とも奇数の場所は絶対に道になり、両方とも偶数の場所は絶対に壁になるとします。

3.3.3 壁を作る (開始点が壁)

```
    void make_wall_A(MAP *map,int y,int x){
        int now,old;

        now = rand()%4;

        while(1){
            if( y + 2*vec[now].y >= 0 && x + 2*vec[now].x >= 0 &&
            y + 2*vec[now].y < map->height && x + 2*vec[now].x < map->width){
                if(map->map[y + 2*vec[now].y][x + 2*vec[now].x] == NONE){
                    map->map[y + vec[now].y][x + vec[now].x] = WALL;
                    y += vec[now].y;
                    x += vec[now].x;
                }else break;
            }else break;
            if(!(x%2 || y%2)){
                old = now;
                now = rand()%4;
                while((now + 2)%4 == old) now = rand()%4;
            }
        }
    }
```

変数 now には現在の進行方向を格納しておきます。進行方向 2 つ先のマスが壁になるまで目の前を壁に変えながら進みます。縦横座標が両方とも偶数になる度に方向転換をします (後ろには戻りません)。

3.3.4 壁を作る (開始点が道)

```
void trans_block(MAP *map,char src ,char dest){
    int y,x;
    for(y=0;y < map->height ;++y){
        for(x=0; x< map->width; ++x){
            if(map->map[y][x] == src) map->map[y][x] = dest;
        }
    }
}

void make_wall_B(MAP *map,int y,int x){
    int now,old;

    now = rand()%4;
    map->map[y][x] = TEMP;
    while(1){
        if( y + 2*vec[now].y >= 0 && x + 2*vec[now].x >= 0 &&
        y + 2*vec[now].y < map->height && x + 2*vec[now].x < map->width){
            if(map->map[y + 2*vec[now].y][x + 2*vec[now].x] == NONE){
                map->map[y + vec[now].y][x + vec[now].x] = TEMP;
                y += vec[now].y;
                x += vec[now].x;
            }else if(map->map[y + 2*vec[now].y][x + 2*vec[now].x] == TEMP){
                trans_block(map,TEMP,NONE);
                break;
            }else{
                map->map[y + vec[now].y][x + vec[now].x] = TEMP;
                trans_block(map,TEMP,WALL);
                break;
            }
        }else {
            trans_block(map,TEMP,NONE);
            break;
        }
        if(!(x%2 || y%2)){
            old = now;
            now = rand()%4;
            while((now + 2)%4 == old) now = rand()%4;
        }
    }
}
```

前項とは逆に、進行方向 2 マス先が壁になるまで 1 マス先を壁に変えながら進行していきます。ただし、すぐに変更するわけでなく、一度 TEMP をいれておき、うまく壁に到達できた場合は WALL に、それ以外なら NONE に trans_block を呼び出して変更します。

3.3.5 終了条件

```
int check_end(MAP *map){
    int y,x;
    for(y=0;y < map->height ;++y){
        for(x=0; x< map->width; ++x){
            if(!(y%2 || x%2) && map->map[y][x] != WALL ) return 1;
        }
    }
    return 0;
}
```

check_end は壁になる可能性のある全てのマス (縦横座標が両方偶数) に対して既に壁になっているか確認する。全て壁になっていれば迷路生成完了。

3.4 完成すると

生成した迷路を表示すると図 3.3 のようになります (71 マス× 71 マス)。

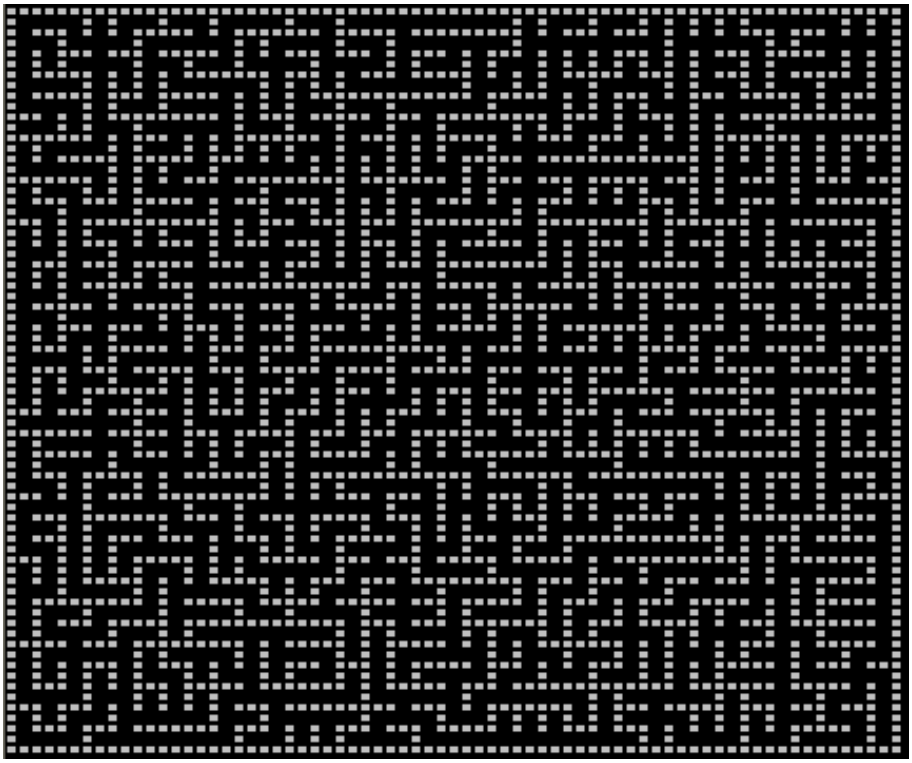

図 3.3: 完成したマップ

3.5 最後に

んー、書けた書けた……1 回生と同レベルだなんてけなさないで〜。とりあえず、迷路生成はこれで終わりです。この作り方は 1 例ですし、手直しするところも多々あります（^^;）興味のある方は色々試してみると面白いかもしれません。とりあえず私は……学園祭の展示品製作に戻ります。果たして無事に迷路は展示品に組み込まれているのか?! 時間との戦いが始まる……。

参考文献

[1] Ishida So
「Samayou Oharikui」
<http://www5d.biglobe.ne.jp/ stssk/>

4 CUIシューを作ってみました

情報工学課程 3 回生 中井 道

4.1 前書き

ネタがありません。やってみたいことはありました。去年はソフトウェアを作ったので今年はなにか実体物を作りたいと思いました。そこで、以前見かけて興味のあった「マウスを用いたペンタブレット[1]」を作ってみようかなと考えました。しかし実現段階までは至りませんでした。他に書きたいことも思い付きません。よって、ネタがない…はずだったのですが、人間不思議なものでいつの間にかキャラクタユーザインタフェース（CUI）を用いてシューティングゲームを作っていました。

さて何故でしょう。事象には必ず原因が存在するものです。この場合、昨年にグラフィカルユーザインタフェース（GUI）を用いてシューティングゲームもどきを作った経験が、いつのまにか私に CUI でのシューティングゲーム開発をさせたのかもしれません。

とまあ、そんなこんなで私は C 言語を用いて CUI でのシューティングゲームを作りました。先に言っておきますと、失敗とまではいかないもののとても良い出来とは思えない作品です。それに、今回は標準ライブラリ関数でないものを用いたので、環境に依存するプログラムになってしまいました。また、実行速度も動作環境に大きく依存しているため、ゲームバランスが崩壊することもあります。だめだめですね。この先を読んで頂ける方はそれを踏まえて読んで頂きたいと思います。

では次の項から、GUI とはまた違った CUI でのシューティングゲームの説明をしたいと思います。

4.2 CUIシューティングゲームの特徴ってなんでしょう

まずは CUI シューティングゲームで特徴的な構造は…うん、特にないですね。構造は基本的に GUI と一緒でした。よく考えたら当たり前のことかもしれません。GUI 環境でも CUI 環境でも同じものを作るなら基本構造は変りません。描画処理が変わる程度です。しかし、肝心の描画処理はむしろ簡単になっています。どうしたものでしょうか…うむ、あまり気にしないことにします。

[1] ここ参照 <http://homepage2.nifty.com/fukanzen/seiei/biboku02/index.html>

4.3 どんなシューティングゲームかな

今回は彗星を迎撃して地球を守る縦シューティングゲームを作成しました。降ってくる彗星をミサイルを用いて破砕することが目的です。一定数見逃したり、自機に衝突すると作戦失敗となり、一定数破砕すると作戦成功となる。そんなシューティングゲームになりました。

4.4 どんな構造になっているのかな

では特に GUI と変わらない構造を説明していこうと思います。

```
メイン関数 {
  無限ループ [1] {
    初期化
    タイトル画面描画
    無限ループ [2] {
      彗星生成
      入力検知および自機移動
      彗星移動
      ミサイル移動
      衝突判定
      画面出力
      終了判定
      実行待機
    }
  }
}
```

全体的な構造はこのようになっています。マップ（移動可能領域）情報は二次元配列で表現しています。配列の添え字は座標を表わし、各値で座標の位置に存在するもの（自機、彗星、ミサイル、何もなし）を表します。

4.4.1 無限ループ [2]

無限ループ [2] では基本的なゲームの実行を行っています。彗星を生成し、入力を検知して自機を移動、彗星やミサイルを移動させ、衝突判定、そして画面出力の後に終了判定をして実行を待機。このような、動作を 1 フレームとし、ループさせることで動いているように描画します。

彗星生成（ミサイル生成）

彗星の生成とミサイルの生成は同じ関数を用いて行います。ただし、彗星が毎フレーム現れると大変なことになるので、乱数を用いて予め定めた頻度で関数を呼び出すこととします。今回は予め生成物の最大数分の構造体配列を用意しておき、有効かどうかを表すメンバ変数を用いることで各

生成物のデータを管理します。この方法だと最大数が大きくなるほどメモリを消費してしまいますので、malloc 関数などを使ってメモリを動的に確保し、リストを生成して管理したほうがよいかと思います。今回は最大数が小さいので、まあ何とかなるでしょう。

```
生成関数{
  ループ（生成物の最大数）{
    判定（構造体配列に空きがある）{
      彗星、ミサイルに共通の初期化処理
      判定 {
        彗星 {
          固有の初期化処理（座標など）
          同座標にすでに存在した場合、データを無効化
        }
        ミサイル { 同様 }
      }
    }
  }
}
```

構造はこのようになっています。

入力検知および自機移動

ここでは conio.h の kbhit 関数、getch 関数という標準ライブラリ関数でないものを用いて入力処理を行いました。環境依存だらけです。kbhit 関数というのはキーボード入力があったかを調べるもので、getch 関数は入力されたキーの値を返すものです。そして、その値に従って、左右移動（今回は上下には動けない）、ミサイル発射の処理を行っています。左右移動は自機の構造体のメンバ変数を変化させます。ミサイル発射は、前項の生成関数を乱数を用いずに呼び出します。

彗星（ミサイル）移動

彗星とミサイルの移動処理は生成関数のように同様の関数を用います。生成物の構造体配列の中から有効であるデータに関して、y 座標を彗星は下に、ミサイルは上に移動させます。しかし、毎フレームごとに移動させたのでは移動が速すぎますので、3 回に 1 回など指定した頻度で移動させるようにしています。

衝突判定

昨年の GUI でのシューティングゲームのときは物体と物体の衝突判定やレーザー光線のような直線と物体の衝突判定など色々と考えた覚えがありますが、今回は非常に簡単です。ただ、座標が重なっているかを確認すればいいのです。ただし、彗星とミサイルが移動時にすれ違う可能性もあ

るので、実際にはミサイルの y 座標が彗星より上であれば衝突したと判定します。x 座標は等しいことが前提ですが。

有効な彗星を有効なミサイル及び自機と座標が重なってないかを判定します。そして、彗星及びミサイルは画面外に出たかどうかも判定します。彗星が画面外に出るということは、地球に直撃するということになります。

画面出力

画面への出力は、まず system 関数に”cls”を渡して画面を全消去します。次にマップ情報の配列について、全座標をいったん何もない空間の値に初期化します。そして、自機、彗星、ミサイルの存在する座標の値を変えたのち、配列の値に応じた記号を描画します。

終了判定

終了の判定は彗星を一定数迎撃したか、自機が一定数彗星の直撃を受けたか、一定数の彗星を見逃したか（地球に直撃）を判定基準とします。終了した場合、無限ループ [2] を抜けます。

実行待機

これは少なくするほど描画間隔が短くなります。for ループを回すことで待機時間を作りました。たぶん、実行環境によってかなり待機時間が異なると思います。また環境依存ですね。

4.4.2 無限ループ [1]

無限ループ [1] では初期化、タイトル画面描画の後、上記の無限ループ [2] を行っています。これでゲームクリアもしくはゲームオーバーになってもタイトル画面に戻れるようになりました。

4.5 実行画面はどうなのさ

実際の実行画面は図 4.1 のようになりました。

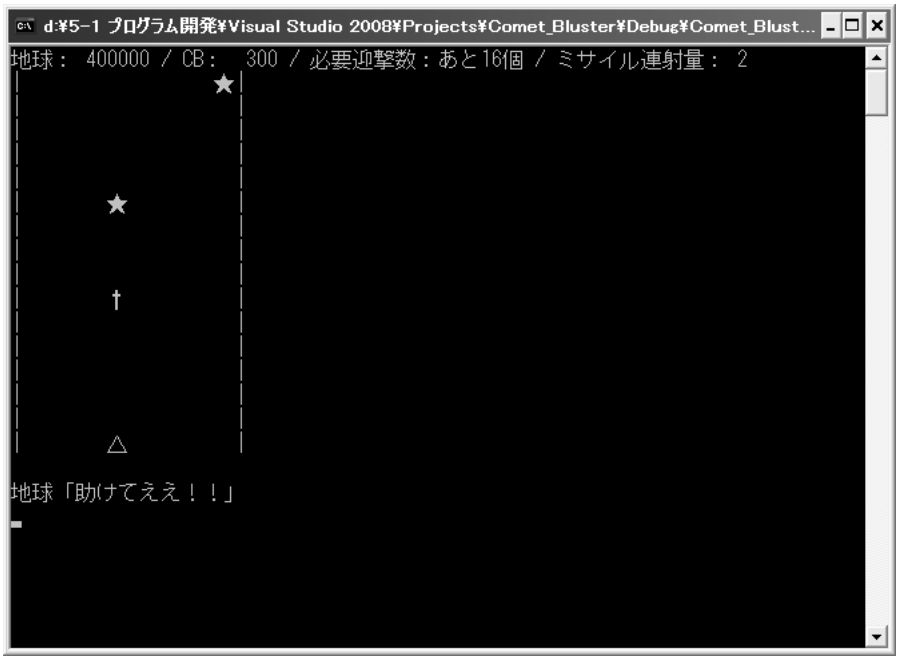

図 4.1: 実行画面

4.6 後書き

今回作成したプログラムの大きな問題点は、環境依存だらけであることと、昨年の作った GUI でのシューティングゲームの時と違い、ダブルバッファリングを用いていないので画面がちらつくことです。とても目によろしくないゲームになりました。

結局のところ、シューティングゲームを CUI で作る必要はあったのかはわかりません。別に GUI でいいような気もします。でも作ることそのものに意義があったと思いますし、なにより楽しかった。それで十分な気がします。

では、このような内容を最後まで読んでくださった方、後書きだけ読んでくださった方…は、いらっしゃらないでしょうね、どうもありがとうございました。最後に本プログラムのソースコード（軽量版）を掲載したいと思います。

4.7 ソースコード

本プログラムからタイトル画面やセリフなどを省いた軽量版のソースコードを掲載します。

```
 1  //CometBlusterLite.c
 2  //彗星迎撃ゲーム「コメットブラスター」
 3  //Lime 用軽量版（タイトル画面、セリフ省略）
 4  //地球を守りましょう。
 5  #include <stdio.h>
 6  #include <conio.h>
 7  #include <stdlib.h>
 8  #include <time.h>
 9
10  #define MAP_X        10  //横幅
11  #define MAP_Y        16  //縦幅
12  #define SPACE        0   //記号：空間
13  #define CB           1   //記号：自機
14  #define COMET        2   //記号：彗星
15  #define MISSILE      3   //記号：ミサイル
16  #define BOMB         4   //記号：爆発
17  #define MAX_COMET    4   //彗星同時出現数
18  #define MAX_MISSILE 2   //ミサイル同時出現数
19  #define SPEED_M      2   //ミサイル速度
20  #define SPEED_C      3   //彗星速度
21  #define FREQ         20  //彗星出現頻度
22  #define NORM         20  //必要迎撃数
23  #define EARTH_LIFE   5   //地球の耐久力
24  #define CB_LIFE      3   //自機の耐久力
25  #define INTERVAL     6000000 //実行間隔
26
27  //表示記号
28  char *display_char[5]={"  ","△","★","†","※"};
29  int map[MAP_Y][MAP_X];     //ステージ領域
30  int earth;       //地球の現在体力
31  int score;       //現在迎撃数
32  //個体パラメータ構成
33  typedef struct object{
34    int exist;     //存在するかどうか
35    int x;         //x 座標
36    int y;         //y 座標
37    int armor;     //耐久力
38    int speed;     //速度
39    int time;      //移動間隔
40  }obj;
41
42  obj cb;                     //自機
43  obj comet[MAX_COMET];       //彗星
44  obj missile[MAX_MISSILE]; //ミサイル
45
46  //衝突判定処理
47  void collisionDetection(void){
48    int i,j;
```

```
49    for(i=0;i<MAX_COMET;++i){
50      if(comet[i].exist==1){
51        if(comet[i].x==cb.x && comet[i].y>=cb.y){
52          comet[i].exist=0;
53          if(--cb.armor<=0){
54            cb.exist=0;
55            return;
56          }
57        }
58        if(comet[i].y>=MAP_Y){
59          comet[i].exist=0;
60          --earth;
61        }
62        for(j=0;j<MAX_MISSILE;++j){
63          if(missile[j].exist==1){
64            if(comet[i].x==missile[j].x && comet[i].y>=missile[j].y){
65              missile[j].exist=0;
66              if(--comet[i].armor<=0){
67                comet[i].exist=0;
68                ++score;
69              }
70
71            }
72          }
73        }
74      }
75    }
76    for(j=0;j<MAX_MISSILE;++j){
77      if(missile[j].exist==1 && missile[j].y<=0)missile[j].exist=0;
78    }
79  }
80  //ミサイル・彗星生成処理
81  void create(obj *c,int n,int s){
82    int i,j,temp;
83    for(i=0;i<n;++i){
84      if(c[i].exist==0){
85        c[i].exist=1;
86        c[i].armor=1;
87        c[i].speed=s;
88        c[i].time=0;
89        switch(n){
90          case MAX_COMET:
91            temp=rand()%MAP_X;
92            if(map[0][temp]==COMET)c[i].exist=0;
93            c[i].y=0;
94            c[i].x=temp;
95            break;
96          case MAX_MISSILE:
```

```
 97            for(j=0;j<n;++j){
 98              if(c[j].exist==1&&i!=j&&c[i].x==c[j].x&&c[i].y==c[j].y){
 99                c[i].exist=0;
100                break;
101              }
102            }
103            c[i].x=cb.x;
104            c[i].y=cb.y;
105            break;
106        }
107        break;
108      }
109    }
110  }
111  //移動処理
112  void move(obj *c,int n){
113    int i;
114    for(i=0;i<n;++i){
115      if(c[i].exist==1){
116        ++c[i].time;
117        if(c[i].time>c[i].speed){
118          c[i].time=0;
119          switch(n){
120            case MAX_COMET:
121              ++c[i].y;
122              break;
123            case MAX_MISSILE:
124              --c[i].y;
125              break;
126          }
127        }
128      }
129    }
130  }
131  //入力判定処理
132  void control(void){
133    int c=-1;
134    if(_kbhit()!=0){
135      c=_getch();
136      switch(c){
137        case'4':if(cb.x>0)cb.x-=1;break;//LEFT
138        case'6':if(cb.x<MAP_X-1)cb.x+=1;break;//RIGHT
139        case'z':create(missile,MAX_MISSILE,SPEED_M);break;//SHOT
140        case 27:exit(EXIT_SUCCESS);//ESC
141      }
142    }
143  }
144  //初期化処理
```

```
145 void init(void){
146   int i;
147   cb.exist=1;
148   cb.x=(MAP_X-1)/2;
149   cb.y=MAP_Y-1;
150   cb.armor=CB_LIFE;
151   for(i=0;i<MAX_COMET;++i)comet[i].exist=0;
152   for(i=0;i<MAX_MISSILE;++i)missile[i].exist=0;
153   earth=EARTH_LIFE;
154   score=0;
155   system("cls");
156 }
157 //コンソール出力処理
158 void display(){
159   int i,j;
160   system("cls");
161   printf("地球：%7d / CB：%5d / 必要迎撃数：
162     "あと%d 個 / ミサイル連射量：%d\n"
163     ,earth*100000,cb.armor*100,NORM-score,MAX_MISSILE);
164   for(j=0;j<MAP_Y;++j){
165     for(i=0;i<MAP_X;++i){
166       map[j][i]=SPACE;
167     }
168   }
169   for(i=0;i<MAX_COMET;++i){
170     if(comet[i].exist==1){
171       map[comet[i].y][comet[i].x]=COMET;
172     }
173   }
174   for(i=0;i<MAX_MISSILE;++i){
175     if(missile[i].exist==1){
176       map[missile[i].y][missile[i].x]=MISSILE;
177     }
178   }
179   if(cb.armor<=0) map[cb.y][cb.x]=BOMB;
180   else map[cb.y][cb.x]=CB;
181   for(j=0;j<MAP_Y;++j){
182     printf("|");
183     for(i=0;i<MAP_X;++i){
184       printf("%s",display_char[map[j][i]]);
185     }
186     printf("|\n");
187   }
188   printf("\n");
189 }
190 //テキスト出力処理
191 void printText(char *t){
192   printf("%s",t);
```

```
193     while(_getch()!=13);
194   }
195   //勝敗判定処理
196   int check(void){
197     if(cb.exist==0){
198       printText("GAMEOVER \t\t\t\t\t\t[↓]\n");
199       return 1;
200     }
201     else if(earth<=0){
202       printText("GAMEOVER \t\t\t\t\t\t[↓]\n");
203       return 1;
204     }
205     else if(score>=NORM){
206       printText("GAMECLEAR \t\t\t\t\t\t[↓]\n");
207       return 1;
208     }
209     return 0;
210   }
211   //メイン処理
212   void main() {
213     unsigned int i;
214     srand((unsigned)time(NULL));
215     while(1){
216       init();
217       while(1){
218         if(rand()%FREQ==0){
219           create(comet,MAX_COMET,SPEED_C);
220         }
221         control();
222         move(comet,MAX_COMET);
223         move(missile,MAX_MISSILE);
224         collisionDetection();
225         display();
226         if(check()==1)break;
227         for(i=0;i<INTERVAL;++i){};
228       }
229     }
230   }
231
```

参考文献

[1] もりじょう
「ささら庵」
<http://www.sasaraan.net/index.html>

5 一晩で作る Lisp

京都大学 工学部 情報学科 2 回生 森下 耕平

5.1 はじめに

この記事は表題にあるように、一晩で Lisp interpreter を作った過程を書いたものです。

5.2 データ関連

まず最初に cons を定義しました。cons を定義する方法はいくつかあるそうですが、今回は共用体を使う事にしました。具体的にはこんな感じです。

```
union cr{
  struct{
    u_int gc   : 1;
    u_int tag  : 2;
    u_int data : 29;
  }data;
  struct cons *cons;
  u_int raw;
};

struct cons{
  union cr car;
  union cr cdr;
};
```

CAR 部、CDR 部の下の 3bit はデータ型を表すタグと、GC（ごみ集め）のためのマーク情報を入れるのに使います。

cons を格納する領域を配列で確保します。そしてこの配列の先頭を指すポインタ cell を作り、cell の指しているアドレスが 8 の倍数でない時は 8 の倍数のアドレスを指すようにずらします。

symbol の名前を格納する領域を char 型の配列で確保します。そしてそれぞれの名前の先頭を指すポインタの配列 symbol_table を作ります。symbol_table のインデックスが、symbol を識別する値になります。

freelist を作ります。freelist は、未使用の cons をリストとしてつないだものです。初期化の際は、free_list の先頭は cell と同じアドレスに設定し、全ての cons は 1 つ後ろの cons を指すようにしま

す（最後の 1 つには NIL を入れます）。

make_cons 関数を作ります。make_cons は freelist の先頭の cons を返し、freelist の先頭を 1 つ後ろにずらします。

intern 関数を作ります。intern は文字列を受け取り、すでにその名前の symbol が存在する場合はその symbol のインデックスを返し、存在しない場合はその名前を新しく格納し、symbol_table に登録してそのインデックスを返します。

5.3 入出力

parse 関数を作り、入力された S 式をリスト、symbol または数に直します。

ここで表示のための関数、print を作りました。

5.4 評価

連想リストの形で環境を作りました。

ついに評価を行う関数、eval を作りました。eval は atom の評価を行い、list に対しては apply 関数を呼び出すようにしています。

apply 関数を作り始めました。apply は入力の第一引数を調べ、それが組み込み関数を示している場合は組み込み関数の処理を行います。第一引数が lambda の場合は関数を表すオブジェクトを作ります。そのオブジェクトは作る関数の仮引数のリストと作る関数本体と lambda の式が評価された時点の環境へのポインタを保持します。

第一引数が関数を表すオブジェクトだった場合は実引数を評価し、仮引数と対応させた連想リストを作り、その連想リストの末尾にオブジェクトが保持している環境をつなげます。こうして作られた新たな環境のもとで関数本体を評価します。

5.5 GC

最後に GC を行う関数を作りました。GC の為に使用中の cons のアドレスを格納する配列 mark_point を用意しました。cons が足りなくなると、GC を行う関数が呼び出され、mark_point に含まれている cons と、そこから参照される cons にマークをつけます。そして全ての cons をチェックして、マークがついていないものを freelist につなぎます。

5.6 おわりに

下は出来たものの実行例です。H E List Processor は「一晩で出来た似非リストプロセッサ」の略です。

```
$ ./help
```

```
H E List Processor
help> (defun f (n) (cond ((eq n 0) 0) (t (+ (f (- n 1)) n))))
f
help> (f 10)
55
```

色々とおかしいところはありますが、なんとか動くものが出来上がってほっとしました。今回は制作時間の都合上、諦めた事や妥協した点があったので、次回はそれを改善したものを作りたいです。

6 エフェクター？いや、エフェクタ作ったぜ

デザイン経営工学課程 2 回生 中野 秀規

6.1 エフェクタって？

そもそもエフェクタとは何か、知らない人はそれほど少なくないと思います。エフェクタとは電子楽器[1]の音に効果を与える機器です。電子楽器では、楽器で音を電気信号に変換し、アンプなどのスピーカーで電気信号を再び音に変換して人の耳に聞こえるようにしています。エフェクタは変換された電気信号が音に変わるまでの所に接続されて、効果を与えます。つまり、楽器とスピーカーの間に接続されるわけです。

では、エフェクタはどんな効果を与えるのでしょうか？その前に音について考えてみます。

6.2 音って？効果って？

音は空気の振動です。つまり波です[2]。波なので、振幅、振動数、波形、速度などの要素があり、その中でも振幅、振動数、波形はそれぞれ音量、音程、音色という音の三要素に大きく関わっています。この三つを変化させることにより、エフェクタは音に効果を与えているのです。

効果には様々な種類があり、振幅を変化させたり、一定レベルに抑制したりする音量系、時間を遅らせて反響した音になる時間系、特定周波数帯の音を制御するイコライザ系などなどさまざまですが、今回は歪み系、その中でもオーバードライブという、非常にポピュラーなエフェクタを製作しました。

6.3 エフェクタの中身

今回のエフェクタは**石崎工房さんの HAM-01S** というエフェクタをもとにして製作しました。回路図は図 6.1 です。実際に制作した基板の写真も載せておきます。部品側が図 6.2 で、配線側が図 6.3 です。

[1]その中でも今回はエレキギターやエレキベース用のエフェクタに焦点を当てています。

[2]三省堂物理小辞典によると、空間や物体内のある部分での振動や変化が次々に隣りの部分に有限の速度で伝わり，遠くまで及んでいく現象を波または波動というそうです。

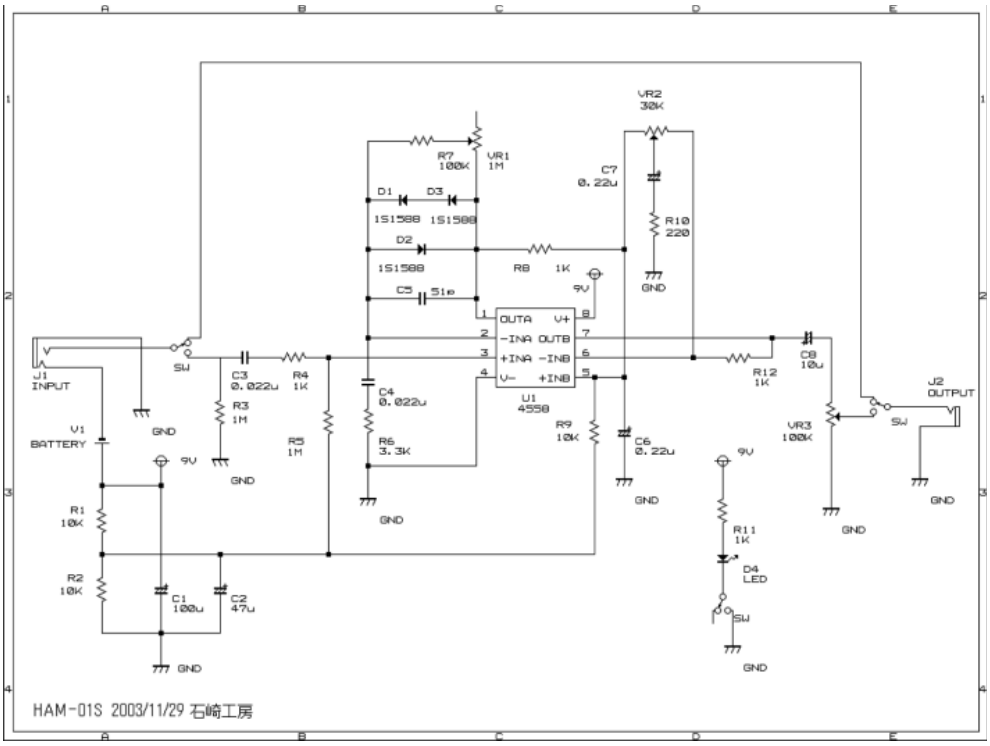
VR2
30K
R7
100K
VR1
1M
C7
0.22u
D1
D3
1S1588
1S1588
R10
220
D2
1S1588
R8
1K
9V
GND
C5
51p
OUTA
V+
-INA
OUTB
+INA
-INB
V-
+INB
U1
4558
J1
INPUT
SW
C3
0.022u
R4
1K
C4
0.022u
R12
1K
C8
10u
J2
OUTPUT
V1
BATTERY
GND
R3
1M
R5
1M
R6
3.3K
R9
10K
C6
0.22u
VR3
100K
SW
9V
GND
R1
10K
R2
10K
C1
100u
C2
47u
R11
1K
D4
LED
SW
GND
HAM-01S 2003/11/29 石崎工房

図 6.1: 回路図

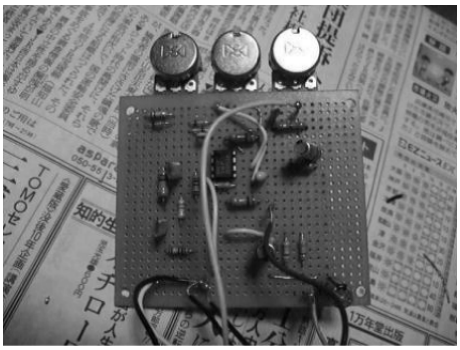

図 6.2: 基板　部品側

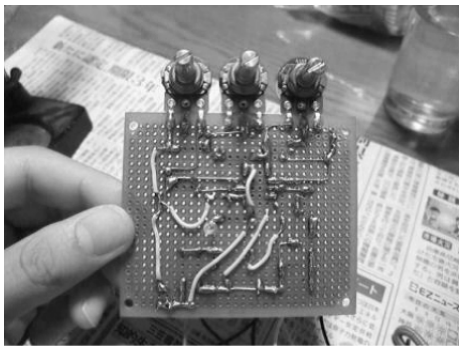

図 6.3: 基板　配線側

それでは、せっかく製作したので使ってみましょう。

6.4 使ってみた

実際にエフェクタを使って音にエフェクトをかけたときと、かけてないときの音を録音し、フリーソフトの **WaveSpectra**[3]を用いて波形を見てみました。

エフェクトをかけていない状態での波形が図 6.4 で、かけている状態での波形が図 6.5 です。

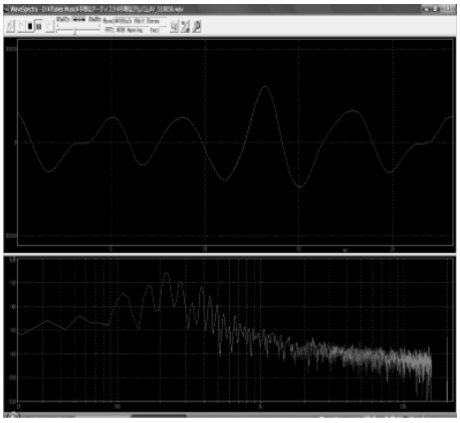

図 6.4: 波形　エフェクトなし

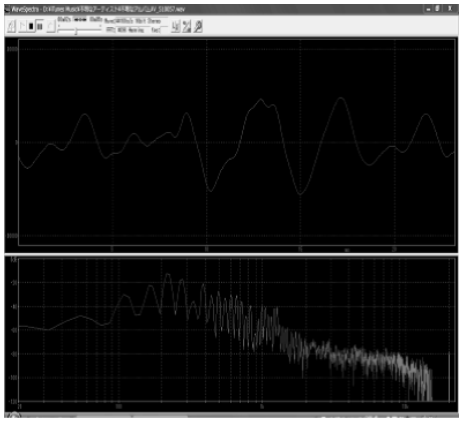

図 6.5: 波形　エフェクトあり

[3]FFT（高速フーリエ変換）により、リアルタイムに周波数成分を表示するツールです。

図 6.4、図 6.5 の波形を見てみましょう。ちなみにどちらも 440Hz の音、つまりラの音で、弦を指で弾いてから 2 秒後の波形です。用いた楽器は僕が持っている Fender 社のプレジションベースです。

エフェクトなしの方は、波形がなめらかで、かなり正弦波[4]に近い形であることが分かります。なのでエフェクトなしでのベースの音は**ポーン**という風に聞こえます。

一方エフェクトありの方は、波の上向きの山の上部と下向きの山の下部がえぐれたような形になっていることが分かります。また、多少ではありますが、振幅が大きくなっているのも分かります。なので音は大きく、**ビィーン**という風に聞こえます。これが歪んだ音です。

また、今回録音した音を**声門**と呼ばれるソフトで声紋画像を表示させてみました。図 6.6 がエフェクトなしで、図 6.7 がエフェクトありです。

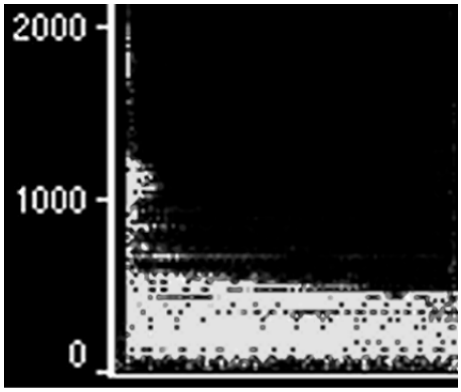

図 6.6: 声紋画像　エフェクトなし

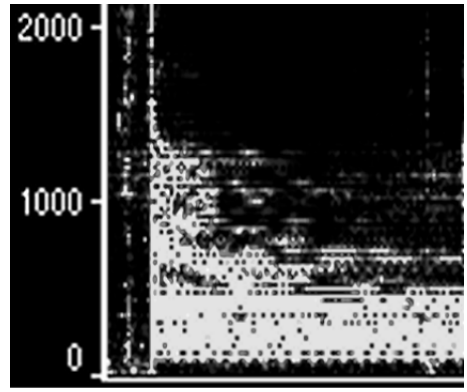

図 6.7: 声紋画像　エフェクトあり

上図は縦軸が周波数で横軸が時間軸です。そして白黒なので分かりにくいとは思いますが、色が白に近いほど強い音、つまり音量の大きい音が発生しています。ちなみに時間 0 付近で高い周波数の音が発生しているのはノイズなどなので無視してください。

ベースの音は 100〜400Hz がよく出ている音なので、そのあたりの周波数の音は非常に白いです。しかし、エフェクトありの方は 1200Hz ほどまで音が出ています。エフェクトをかけたことによって、こういった高い音が合わさっているのです。

ではどうやってエフェクタはエフェクトをかけているのでしょう？

6.5 エフェクタの原理

まず、楽器から来た音の信号を INPUT から取り込み、二回路入りオペアンプ（NJM4558）によりその信号を増幅します。ここでは単純に振幅が大きくなるので、音が大きくなります[5]。その後、回路図左上にある、ダイオードが三つ並んだ部分で音の信号の上向きの山の上部と下向きの山の下

[4]サイン波のことです。音叉等で発生する最も純粋な音です。
[5]オペアンプの種類によっては歪みが大きいものもあるので、単に音が大きくなるだけではない場合もあります。

部を切り取ります（図 6.8 参照のこと）。これがいわゆるクリッピング回路[6]、今回はその中でも非対称系[7]のクリッピング回路です。

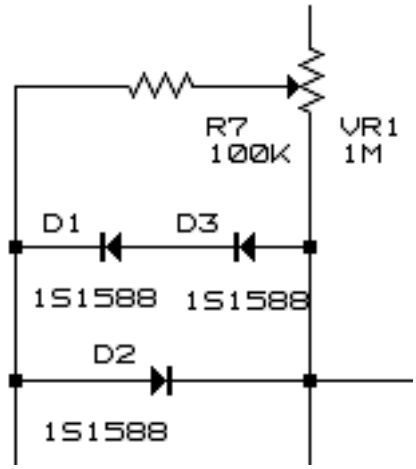

図 6.8: クリッピング回路

6.6 クリッピングの原理

クリッピングの原理を説明するために、まずダイオードの性質を説明します。ダイオードは一方方向にしか電流を流しませんが、ある程度電流が大きくなると、電圧がほぼ一定になります。その電圧が図 6.9 の V_F です。この値はダイオードの種類にもよりますが、今回使ったダイオードではおおよそ 0.6V 程度です。つまり、0.6V 以上はスパッと切られたようになります。これがクリッピングの原理です。

6.7 終わりに

初めての電子工作で、僕の趣味であるベース用のエフェクタを作れて大満足なのですが、アンプは市販のものでした。なのでアンプを、出来ればイコライザとチューナーのついたアンプを作れたらいいな、と考えております。

[6] リミッタ回路とも呼ばれます。
[7] 向きの違うダイオードの片側の個数、または種類が違うことにより、切り取る（クリッピングする）度合いが違うクリッピング回路の事です。

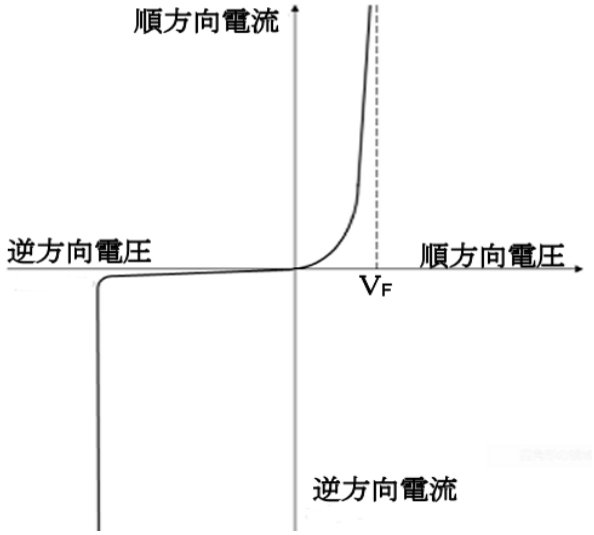

図 6.9: ダイオードの基本性質

参考文献

[1] i-zaki
「石崎工房」
<http://hamsan.ciao.jp/index.html>

[2] 後閑 哲也
『誰でも手軽にできる電子工作入門』 技術評論社

[3] 畑野 貴哉
『土日で作るオリジナルエフェクター』 リットーミュージック

[4] 三省堂編修所
『三省堂物理小辞典』三省堂

7 明日にときめく昆布鉄道　～萌える部員の勇姿を乗せて～

電子システム工学課程 2 回生 渡邉 翔一郎

7.1 ご乗車ありがとうございます

7.1.1 私の電子工作

私が電子工作をするときは、回路の設計→ブレッドボードで実験→回路の書き換え→はんだ付けといった手順を踏みます。シミュレータで済ましてしまうのもひとつの手ですが、次の 7.4.3 の場合のように抵抗を何十本も直列につなぐような場合は各抵抗の誤差も考慮する必要があります。ロット買いするようなカーボン抵抗では誤差が 5 %ほどあるので、こればかりは実際確かめる方が無難なのです。ですから私は面倒でもブレッドボードで実験します。まぁ面倒だなんて思ったことはありませんし、正直な話、画面上で回路を組んでニヤニヤするよりも実際に道具を使ってニヤニヤするほうが好きです。ちなみに私は四国出身ですが、初めて京都の寺町電気街で買ったものがブレッドボードでした。これを買ったときは「ついにオレの時代が来た」と喜びました。でも実際は……

7.1.2 鉄道模型ってな～に？

実際の鉄道車両を忠実にモデル化したものを鉄道模型といいます。この鉄道模型では車両を線路の上で電気でコントロールして走らせたり、その線路の沿線で踏み切りや信号などを動作させて楽しむことができます。鉄道模型には数種類の大きさがありますが、そのうち私は線路幅 9 ミリの「N ゲージ」で遊んでいます。私はこの鉄道模型をもっと楽しくコントロールして遊びたいなぁと思い、はんだごてを握りました。
「ど～せやるからには鉄道会社に勝てるシステムを作ろう！」
図 7.1 は実際の鉄道車両と模型車両の写真です。ぜひ見比べてみてください。

7.2 ご案内いたします

7.2.1 プロジェクト概要

私は以下のような計画を立てました。

図 7.1: 実車/鉄道模型の写真

- マスコン回路
- レベルメータを使った電圧表示
- 最高速度超過警告
- 区間規定速度超過警告
- 電流超過警告・遮断
- 信号機
- 踏み切り
- ポイント切り替え
- トランジスタブザー
- 場内放送再生
- 車載カメラ
- 電光掲示板

これらのうち上 2 つを簡単に紹介します。

7.3 発車いたします！ドアにご注意ください！

マスターコントローラー回路回路の説明

7.3.1 マスターコントローラー

鉄道車両の制御器をマスターコントローラー（マスコン）といいます。鉄道模型でも車両を制御するためにこのマスコンが必要です。マスコンにはひとつのハンドルで操作する 1 ハンドル型と、加速減速それぞれを操作する 2 ハンドル型があります。

7.3.2 マスコンの回路

鉄道模型ではレールから車両に電力を供給します。直流制御と交流制御がありますが、今回は直流で制御する手段をとります。制御の方法はいたって簡単で電源の出力電圧を操作するだけで早さの調整ができます。もっとも、学校の理科室にあるような電源装置をレールにつないで、つまみをまわして電圧を上げ下げすれば速さの制御ができるのです。な〜んだ簡単じゃん。**でも私はこだわってしまいました。**なぜなら実車ではそんな操作をしないからです。（あと私がマニアだからです。）実車では車両の「加速度」を操作します。先述のようにつまみを回して電圧を上げ下げする方法では比例操作で加速度を操作できません。鉄道模型で加速度を操作するには「電圧の大きさ」ではなく「電圧の増減の大きさ」を操作する必要があります。

7.3.3 エミッタフォロア回路

7.3.2 で挙げたこだわりを実現する方法として、私はエミッタフォロア回路というものを利用しました。エミッタフォロア回路とはトランジスタを使った回路の一つで、コレクタ接地回路とも呼ばれます。主な特徴としては

- 電圧増幅率が約一倍
- 出力インピーダンスが低い

が挙げられます。図 7.2 に設計した回路図を載せておきます。

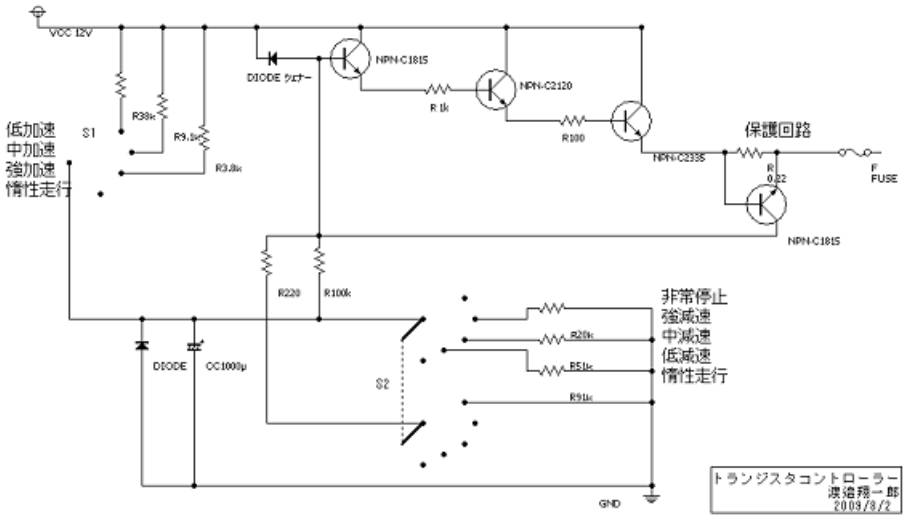

図 7.2: マスコン回路図

7.3.4 回路の説明

トランジスタをトントントンっと 3 個つないでいますが、これには名前がついていて「ダーリントン回路」と呼ばれています。1 段目で増幅して、それをまた増幅して、もういっちょ増幅して…といった流れになります。また、ダーリントン 3 個目のトランジスタは放熱のことも考えなくてはいけません。高電圧で出力しているときはさほど問題になりませんが、低電圧で出力しているときは負荷がかかり結果熱を生み出してしまうのです。

加速度の大きさは加速減速ともに接続される抵抗の値によって決まります。加速はスライド抵抗で抵抗値を変化させ、減速はロータリースイッチで抵抗を選択・切り替えます。

ショートなどで回路が損傷する恐れのある場合には保護回路が機能します。保護回路のトランジスタは強い電流が流れるとコレクタ電流が流れやすくなり、ダーリントン回路 1 段目のベースの電位が下がります。こうなるとダーリントン回路 3 段目のトランジスタに流れるコレクタ電流が制限され、結果回路へのダメージが減ります。また回路図にはありませんが、ヒューズはポリスイッチを使っています。ポリスイッチとは、ある一定の電流より強い電流が流れると小さな抵抗だったものが大きな抵抗へと変化して、事実上電流をシャットダウンする特性があります。

電解コンデンサは容量を変えると加減速の大きさが変化します。今回、加減速は抵抗の値を変えて変化させることにしています。

「非常停止」すると 1 段目のトランジスタのベースの電位が 0 になるので出力はいきなり 0 になります。解除すると非常停止する前の出力電圧に復旧します。

7.3.5 完成基板

完成した基盤は写真のようになりました。3 段目のトランジスタは放熱板をつけるため基板外と接続するピンを立てておきます。おまけに部室での製作風景もどうぞ。

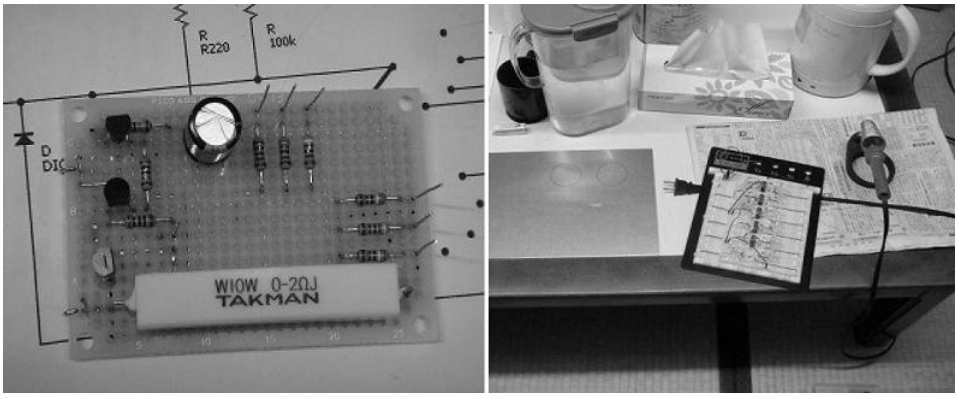

図 7.3: 完成基板/部室での製作風景

7.4 駆け込み乗車はおやめください！ドア閉まります！

レベルメータを使った電圧表示

7.4.1 レベルメータとは？

オーディオコンポやカラオケ機器などで音量を上げていくと緑→黄→赤といった具合に点灯するところを見たことはありませんか。レベルメータとは電圧や音など信号の大きさに比例して点灯するランプの数を増すことでその大きさを測る装置です。今回のプロジェクトではマスコンの出力電圧をこのレベルメータを使って表すことにしました。

7.4.2 コンパレータ

設計した回路図ではコンパレータ IC を使用しています。この IC は名前からも想像できるように入力電圧と基準電圧を比較する IC です。基準電圧より低い入力電圧の場合は出力は GND と等電位になり、逆の場合は Vcc の電圧を出力します。基準電圧を作るために抵抗を直列にいくつかつないで、各々の結節点を引っ張り出します。

7.4.3 回路の説明

工作では 6 個のコンパレータ IC と 24 個の LED を使いましたが、ここでは回路の一部を抜粋して載せます。ここからは図 7.4 の回路図に沿って説明します。コンパレータの IC は LM339 という型番です。1 個につき 4 回路内蔵されています。上記の回路図では入力電圧を+、基準電圧を-に対応させています。

R3〜6 は基準電圧を作る部分です。抵抗を 4 つ直列につないで分圧を作ります。最初に述べましたが、基準電圧を作る抵抗は各々の誤差が数が増える分だけ大きくなるので注意が必要です。そして流れる電流は D1 の定電流ダイオードで 1mA に保ちます。

R3〜5 で基準電圧は 0.1 Ｖ刻みになっています。入力電圧は 0〜12 Ｖを想定していますが、これでは大きすぎるため大きさを 1/10 に抑えて比をあわせます。そのためコンパレータへの入力は R1 を挟んで対応させます。

コンパレータは基準電圧を超えると出力が L、つまり GND と等電位になるので LED はカソード側をコンパレータにつなぎます。

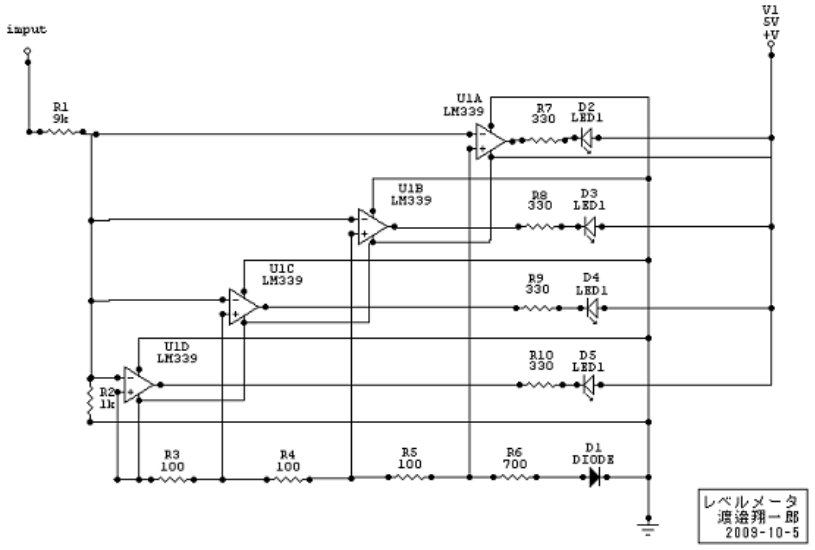

図 7.4: レベルメータ回路図

7.5 ご乗車ありがとうございました。

今回は工作の進捗状況にあわせて原稿を書いていたために本誌で紹介できるプロジェクトが 2 つだけになってしまいました。ですがこの 2 つは私がかなり時間をかけて入念に作った回路です。回路図などを見て「こんな動作するのかぁ〜。おもしろいなぁ〜。」と思っていただければ幸いです。作品が完成したとき、私は「よっしゃぁぁぁああああーーーーー！できたぁぁぁぁぁああああああっ！」と思いっきり部室でガッツポーズして浮いていました。読者の方の中には「これだけで？」と思っておられる方もいるかと思います。ですが私は作品が完成したこと以上に、周りの期待に応えられたのがうれしかったのです。ハード・ソフト問わずトコトン議論してくださる先輩、作業を手伝ってくれる同級生、一緒に作品で遊んでくれる後輩など私は周りに支えられました。この十分すぎるほどに恵まれた環境に私は感謝しています。学祭本番では颯爽と鉄道模型が走り、爽快に皆様がときめき、私の心が快楽に満ちていることを妄想しつつ、次の電子工作に向けて ”発車します！”。

ご拝読ありがとうございました。またのご利用お待ちしております。

参考文献

[1] 丹波山 次郎
『トランジスタ活用はじめの一歩』 CQ 出版社

[2] JR1CPD

「鉄道模型製作のページ」
<http://hw001.gate01.com/cpd/newpage3.htm>

[3] 赤姫
「尾張内燃鉄道」
<http://homepage2.nifty.com/baldwin/>

[4] MasaRi
「The Bipolar Circuits」
<http://bipcircuits.web.fc2.com/index.html>

[5] PC Watch
「武蔵野電波のブレッドボーダーズ　バックナンバー」
<http://pc.watch.impress.co.jp/docs/article/backno/musashino.htm>

8 TCP/IP ネットワークプログラミング

先端科学技術課程 1回生 原 一貴

8.1 はじめに

夏休みの後半になり、そろそろ何か始めないといけないなぁと思い本屋で目についたのが「C 言語実践 Linux システムプログラミング」でした。C 言語の勉強は前期に部活であった C 言語勉強会に参加したり、自分で本「やさしい C」を買って基本的な勉強をしました。そして、Linux システムプログラミングの本で一番面白そうだなと思ったのが TCP/IP の項目でした。

さて TCP/IP の項目を理解して何を作ろうかと考えて多人数が接続できる ECHO プログラムを作ることにしました。今回の Lime にはサーバプログラムを作る際とても興味深い C 言語の使い方をみつけたので、それについて書きます。

8.2 ファイルディスクリプタについて

ファイルディスクリプタというのはプログラムがアクセスするファイルや標準出力などの I/O を OS が識別するために使う識別子で番号が与えられます。0〜2 番については OS のシェルが初めに割り当てます。番号が割り当てられるのでどこの I/O が ready 状態（待機状態）になったかを簡単に見分けることが出来ます。

8.3 select について

select というのはシステムコールで fd_set マスクで指定された I/O(ファイルディスクリプタの集合) を監視し、集合の中のどこかのファイルディスクリプタ (I/O の番号) が ready 状態になるまでプログラムを停止します。なお、timeout を指定してやることで、どれも ready 状態でない場合でも処理を進めることが出来ます。select の返り値は ready 状態となっているファイルディスクリプタの個数です。select を使うにあたって注意があります。それは select に渡したファイルディスクリプタは select 終了後はどうなってるかわからなくなることです。したがって select に入る前に毎回 memcpy で構造体をコピーをする必要があります。select は下記のような形をとります。

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

nfds には監視するファイルディスクリプタの個数を入れます。次に、readfds に読み込み待機になっているかどうか調べたいファイルディスクリプタ集合のアドレスを入れます。writefds, exceptfds

にはそれぞれ書き込み、例外が待機状態になっているかどうか調べたいファイルディスクリプタの集合のアドレスをいれます。最後 timeout は監視を続ける時間（timeval 構造体のアドレス）を入れます。

8.4 fd_set マスクおよび、ファイルディスクリプタに関するマクロについて

fd_set マスクは簡単に言えばファイルディスクリプタをまとめた集合です。しかしファイルディスクリプタが作成されたときに自動で集合に登録される訳ではありません。また、fd_set マスクを用意しても FD_ZERO で初期化する必要があります。

そして、登録を行うマクロが FD_SET です。FD_SET にファイルディスクリプタ番号と fd_set マスクのアドレスを投げてやると指定されたアドレスの fd_set マスクにファイルディスクリプタが登録されます。

次に FD_ISSET です。FD_ISSET は指定されたファイルディスクリプタが fd_set マスクで ready 状態になっているか確認します。ready 状態であれば 1 を、ready 状態でなければ 0 を返します。次に FD_SETSIZE です。これは OS が用意できるファイルディスクリプタの最大数です。OS によってこの値は変わります (Linux および MacOSX は 1024)。

そして FD_CLS です。これは監視対象から外したいファイルディスクリプタを fd_set マスクから外します。これらは下記のような形をとります。

```
void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

fd は監視対象のファイルディスクリプタです。set はファイルディスクリプタの集合のアドレスです。さて、上記で述べたことを実際にプログラムにしまとめてみましょう。

```
Int n;
fd_set fd, org_fd;
FD_ZERO(&org_fd);
n =‘ ファイルディスクリプタを返り値に持つ関数 ’
FD_SET(n, &org_fd);
memcpy(&fd, &org_fd, sizeof(&org_fd));
select(FD_SETSIZE, &fd, NULL, NULL, NULL);
if( FD_ISSET(n, &fd)) )printf(“ %d が ready 状態です。\n ”, n);
```

簡単にまとめてみました。しかし実際は select のエラー処理をしないといけません。

8.5 完成したプログラム:サーバプログラム

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/socket.h>
```

```
 4 #include <netinet/in.h>
 5 #include <arpa/inet.h>
 6 #include <unistd.h>
 7 #include <time.h>
 8 #include <string.h>
 9 #include <sys/time.h>
10 #include <stdlib.h>
11 #include <math.h>
12 #define PORTS 6901
13 #define BUF_SIZE 1024
14 char buf[BUF_SIZE];
15 int listen_fd;
16 int fd_max;
17 struct sockaddr_in server;
18 fd_set readfd;
19 int connect_client();
20 int send_message(int fd);
21 int recv_message(int fd);
22 int do_close(int fd);
23
24 int main (int argc, const char * argv[]) {
25   fd_set rfd;
26   int result;
27   int i;
28   int ret;
29   FD_ZERO(&readfd);
30   listen_fd = socket(AF_INET, SOCK_STREAM, 0);
31   if (listen_fd < 0 || listen_fd >= FD_SETSIZE ) {
32     perror("socket");
33     close(listen_fd);
34     return 0;
35   }
36   server.sin_family = AF_INET;
37   server.sin_port = htons(PORTS);
38   server.sin_addr.s_addr = INADDR_ANY;
39   result = bind(listen_fd,  (struct sockaddr *)&server, sizeof(server));
40   if ( result == -1 ) {
41     perror("bind");
42     return 0;
43   }
44   result = listen(listen_fd, 5);
45   if ( result == -1 ) {
46     perror("listen");
47     return 0;
48   }
49   if (fd_max <= listen_fd) fd_max = listen_fd + 1;
50   FD_SET(listen_fd, &readfd);
51   while (1) {
```

```
52      memcpy(&rfd, &readfd, sizeof(readfd));
53      ret = select(fd_max, &rfd, NULL, NULL, NULL);
54      if(ret < 0 ) {
55        perror("Select");
56        return 0;
57      }
58      for ( i = 0; ret > 0 && i < fd_max; ++i) {
59        if ( FD_ISSET(i, &rfd) ) {
60          if ( i == listen_fd ) {
61            connect_client();
62            --ret;
63            continue;
64          }
65          recv_message(i);
66          send_message(i);
67          --ret;
68        }
69      }
70    }
71    close(listen_fd);
72    return 0;
73  }
74  int connect_client()
75  {
76    int fd;
77    struct sockaddr_in client;
78    socklen_t len;
79    len = sizeof(client);
80    fd = accept(listen_fd, (struct sockaddr *)&client, &len);
81    if (fd < 0) {
82      perror("accept");
83      return 0;
84    }
85    if ( fd_max <= fd ) fd_max = fd + 1;
86    FD_SET(fd, &readfd);
87    printf("acceptOK fd: %d\n", fd);
88    return 0;
89  }
90  int recv_message(int fd)
91  {
92    int len;
93    len = recv(fd, buf, BUF_SIZE, 0);
94    if ( len < 1 ) {
95      do_close(fd);
96      return 0;
97    }
98    return 0;
99  }
```

```
100 int send_message(int fd)
101 {
102   int len;
103   char buf_len;
104   buf_len = buf[0];
105   len = send(fd, (const char *) buf, (int)buf_len, 0);
106   if ( len <= 0 ) {
107     do_close(fd);
108     return 0;
109   }
110   return 0;
111 }
112 int do_close(int fd)
113 {
114   close(fd);
115   FD_CLR(fd, &readfd);
116   return 0;
117 }
```

8.6 クライアントプログラム

```
 1 #include <stdio.h>
 2 #include <sys/types.h>
 3 #include <sys/socket.h>
 4 #include <netinet/in.h>
 5 #include <arpa/inet.h>
 6 #include <unistd.h>
 7 #include <time.h>
 8 #include <string.h>
 9 #include <sys/time.h>
10 #include <stdlib.h>
11 #include <math.h>
12 #include <netdb.h>
13 #include <netinet/tcp.h>
14 #define PORTS 6901
15 #define BUF_SIZE 1024
16 int do_close(int fd);
17 fd_set readfd;
18 int sock;
19 struct sockaddr_in server;
20 char readbuf[BUF_SIZE], sendbuf[BUF_SIZE];
21 int main (int argc, char * argv[]) {
22   fd_set rfd;
23   char *deststr;
24   size_t len;
25   char body[200];
26   int flag = 1;
27   if (argc != 2) {
28       printf("Usage : %s dest\n", argv[0]);
```

```
29          return 1;
30      }
31      deststr = argv[1];
32      sock = socket(AF_INET, SOCK_STREAM, 0);
33      server.sin_family = AF_INET;
34      server.sin_port = htons(PORTS);
35      if (server.sin_addr.s_addr == 0xffffffff) {
36          struct hostent *host;
37          host = gethostbyname(deststr);
38          if (host == NULL) {
39              return 1;
40      }
41          server.sin_addr.s_addr = *(unsigned int *)host->h_addr_list[0];
42      }
43      connect(sock, (struct sockaddr *)&server, sizeof(server));
44      FD_ZERO(&readfd);
45      FD_SET(sock, &readfd);
46      FD_SET(0, &readfd);
47      while(1) {
48          memcpy(&rfd, &readfd, sizeof(readfd));
49          if(select(FD_SETSIZE,&rfd,NULL,NULL,NULL) < 0) {
50              perror("Select");
51          }
52          if (FD_ISSET(0, &rfd)) {
53              fgets((char *)body, sizeof(body), stdin);
54              len = strlen((char *)body);
55              body[len-1] = '\0';
56              sendbuf[0] = len + 1;
57              memcpy((sendbuf + 1), body, len);
58              if( write(sock, sendbuf, len + 1) < 1) {
59                  perror("write");
60                  do_close(sock);
61              }
62          }
63          if (FD_ISSET(sock, &rfd)) {
64              if ( read(sock, readbuf, sizeof(readbuf)) < 0 ) {
65                  perror("read");
66                  do_close(sock);
67              }
68              printf("[length: %d]: %s\n", *readbuf[0], (readbuf + 1));
69          }
70      }
71      close(sock);
72      return 0;
73  }
74  int do_close(int fd)
75  {
76      close(fd);
```

```
77    FD_CLR(fd, &readfd);
78    return 0;
79  }
```

8.7 最後に

今回、ECHO プログラムを書くにあたり man にはとてもお世話になりました。ここでお礼申し上げます。これから今回つくった ECHO プログラムを発展させゲーム的要素を持たせたチャットプログラムを作ろうと考えています。また、コマンド機能も実装していこうと考えています。

参考文献

[1] 小俣 光之
『C 言語による実践 Linux システムプログラミング』 秀和システム，2007

[2] 高橋 麻奈
『やさしい C』 第 3 版 SoftbankCreative, 2007

[3] 小俣 光之
『C 言語による TCP/IP ネットワークプログラミング』 ピアソン・エデュケーション，2003

9 STG(シューティングゲーム)を作ってみた。

情報工学課程 1 回生 高岡 勇紀

9.1 はじめに

コンピュータ部の勉強会で C 言語の基本を習い、○×ゲームや五目並べといったテーブルゲームは作れるようになりましたが、もっとゲームらしいゲームをと思い、簡単な STG を作ってみることにしました。STG を作るといってもどうすればいいのか全く分からないので、とりあえず、参考書がほしいと思い、買ったのが「14 歳からはじめる C 言語わくわくゲームプログラミング教室」という本です。この本を参考にして作っていきます。

9.2 内容

今回製作した STG は山田巧氏が作成した、DX ライブラリというゲーム用ライブラリを用いて C 言語で記述しました。ライブラリの使い方を覚えるとともに、よい C 言語の復習にもなりました。この記事では簡易 STG を作るまでの簡単な手順を書いていきたいと思います。

9.2.1 キャラクターを動かす

まずは DX ライブラリでキャラクターを表示してみましょう。画像をファイルから主記憶上に読み込みます。LoadGraph 関数を使います。

```
int LoadGraph( char *FileName );
```

第一引数の FileName で読み込み元のファイルを指定します。BMP のほかに、JPEG、PNG、DDS、ARGB、TGA に対応しているようです。戻り値は読み込んだ画像の識別番号です。後で画像をスクリーンに貼り付けたり、主記憶上から解放したりするときに使用します。次に読み込んだ画像を表示させます。DrawGraph 関数を使います。

```
int DrawGraph(int x,int y,int GrHandle,int TransFlag);
```

第一、二引数 x,y で表示させたい画像の左上の座標を指定し、第三引数の GrHandle に先ほど読み込んだ画像の識別番号をいれます。第四引数の TransFlag を TRUE にすることで透過色を有効にします。FALSE の場合、透過色が無効になります。これでキャラクターは表示できました。次に動かします。ClsDrawScreen 関数、SetDrawScreen 関数、ScreenFlip 関数を使います。

```
  int ClsDrawScreen(void);
  int SetDrawScreen(int DrawScreen);
  int ScreenFlip(void)
```

画像を動かすということは画像の位置を少しずつ動かしながら、繰り返し表示することです。この時、ただ画像の表示する座標を変えるだけでは表示した画像の上に新しい画像を重ね書きしてしまい、動いた跡が残ります。ClsDrawScreen 関数は画面を消去する関数です。この関数により動いた跡を消すことができます。また、DX ライブラリには通常の画面の他にウラ画面が用意されていて、SetDrawScreen 関数で書き込む対象を指定できます。第一引数の DrawScreen で DX_SCREEN_FRONT を指定するとオモテ画面、DX_SCREEN_BACK を指定するとウラ画面となります。ScreenFlip 関数を実行するとウラ画面とオモテ画面が入れ替わります。ウラ画面に画像を書き込み、書き込みが終わってからオモテ画面と入れ替えれば、書き換えている途中は見えなくなります。
上記を踏まえてソースを以下のように書きました。これでなんとかキー入力に応じてキャラクターを動かすことができました。

```
int WINAPI WinMain(HINSTANCE hI, HINSTANCE hp, LPSTR lpC, int nC)
{
  ChangeWindowMode(TRUE);           //ウィンドウモードで起動
  if(DxLib_Init() == -1)return(-1);//DX ライブラリ初期化
  SetDrawScreen(DX_SCREEN_BACK);
  image = LoadGraph("image.bmp");
  while(ProcessMessage() == 0 && CheckHitKey(KEY_INPUT_ESCAPE) == 0 ){
    ClsDrawScreen();
    //キャラクター移動
    int key = GetJoypadInputState(DX_INPUT_KEY_PAD1);
    if(key & PAD_INPUT_UP & y > 0)      y=y-4;
    if(key & PAD_INPUT_DOWN & y < 433)  y=y+4;
    if(key & PAD_INPUT_LEFT & x > 0)    x=x-4;
    if(key & PAD_INPUT_RIGHT & x < 592) x=x+4;
    DrawGraph(x, y, image, TRUE);
    ScreenFlip();
  }
  DxLib_End();
  return(0);
}
```

9.2.2 弾

では次に先ほど作ったキャラクターから弾を発射させたいと思います。弾は直線軌道で、発射キーが押されたとき弾を表示し、それを画面端まで移動させます。複数の弾を発射させたいので弾関連の処理を for 文で繰り返させます。これで弾を発射することができます。コンパイルして実行すると弾は出ますが発射間隔が短すぎて重なって見えます。発射間隔を広げる工夫が要るようです。変数を追加してその数値を変えることで発射間隔を調節できるようにしました。そして、弾幕系 STG の自機の弾のような弾速及び発射間隔にしました。

9.2.3 当たり判定

最近の STG の当たり判定は自機（自分の動かすキャラクター）の図の中心に円があり、その内側に判定領域（判定に使用する領域）があるものをよく見かけますが、今回は長方形で判定します。読み込んだ図の高さの 2 分の 1 を image_h_half、幅の 2 分の 1 を image_w_half とし、当たり判定に使う一回り小さめの四角形の高さを bounds_h_half、幅を bounds_w_half として数値を入れることにしました。下は a と b の二つのキャラの判定方法です。8 つの式で判定領域を求めて、if 文で判定しています。

```
int IsAtari(CharaData a, CharaData b){
  int retval = 0;
  int ax1 = a.x + a.image_w_half - a.bounds_w_half;
  int ay1 = a.y + a.image_h_half - a.bounds_h_half;
  int ax2 = a.x + a.image_w_half + a.bounds_w_half;
  int ay2 = a.y + a.image_h_half + a.bounds_h_half;
  int bx1 = b.x + b.image_w_half - b.bounds_w_half;
  int by1 = b.y + b.image_h_half - b.bounds_h_half;
  int bx2 = b.x + b.image_w_half + b.bounds_w_half;
  int by2 = b.y + b.image_h_half + b.bounds_h_half;
  if( (ax1<bx2) && (bx1<ax2) && (ay1<by2) && (by1<ay2)) retval = 1;
  return(retval);
}
```

9.3 とりあえず

キャラクターの動作、弾、当たり判定ができたので敵を作りちょっと遊べるようにしてみました。図 9.1 は本記事を書いた時点でのプレイ画像で、少し見にくいですが●が敵です。倒した敵の数（COUNT）と自機の LIFE が表示されています。敵が左向きに流れ、画面端にきたら復活し、右に戻ってまた現れるを繰り返します。見た目には横スクロール風の簡易 STG にしました。動作確認は良好で敵に当たれば LIFE が減り、LIFE が 0 になると GAMEOVER の文字が表示されます。

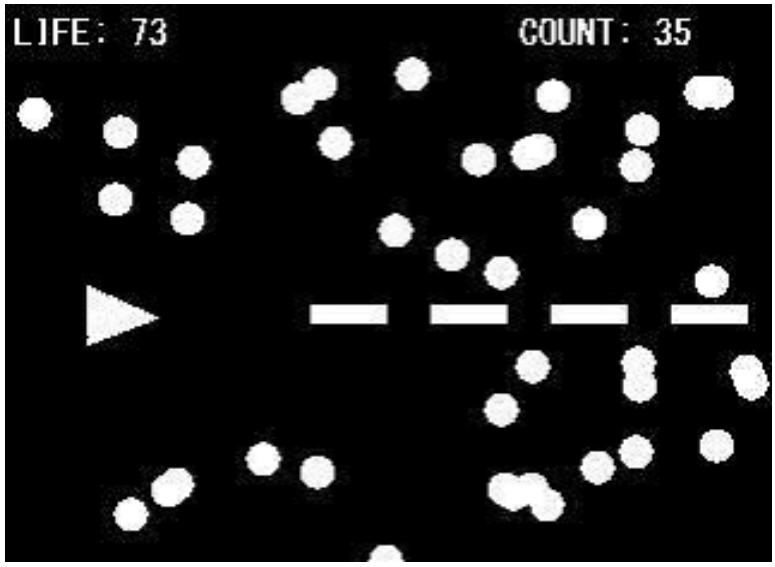

図 9.1: 出来上がり図

9.4 これからの課題

今回は参考にした本の通りに進めたのでオリジナル性はほとんどありません。これから変更を加えてもう少しオリジナル性を出した縦 STG にしたいと思っています。とくに理由はありませんが、なんとなく縦で行きたいと思います。それと、今はまだ敵が一種類しか出せてませんがボスキャラクターも作りたいです。さらにスクロールさせて数面用意したいです。かなり思いつきでやりたいことを書いてますが、頑張って作っていきたいと思います。

参考文献

[1] 大槻 有一郎
『14 歳からはじめる C 言語わくわくゲームプログラミング教室』 株式会社ラトルズ

[2] 山田 巧
「DX ライブラリ置き場」
<http://homepage2.nifty.com/natupaji/DxLib/>

10 C言語とシューティング

電子システム工学課程１回生 松井 隆明

10.1 ゲームをつくろう

ここ最近 C 言語の勉強をしてきた私は、C の基礎でもってなにかプログラムを、楽しめるようにゲームを作ってみようと考えました。そして、シューティングゲームはゲームプログラミングをするにあたって基礎的な部分が多く含まれている、といったことを聞き、ではそれにしようということでＳＴＧの製作に挑戦したわけです。

10.1.1 C 言語

今回のプログラム言語は前述のとおり使用人口の多さに定評のある C 言語です。C はもともと Unix を開発するために作られましたが、やがてその他の OS にも普及し幅広い分野で使われるようになりました。C はソースコードを関数に分割して書く構造化言語の代表格です。

構造化言語の後で登場したオブジェクト指向言語の一つにたとえば Java があります。Java は C よりコードの管理の仕方が改良されわかりやすく書けますが、C のほうがルールが少ないので覚えやすいのです（そうでないという人もいる）。C はソースファイル中に自分で作った関数の他にライブラリ（複数のプログラムをまとめたもの）を取り込むことで使える関数を増やせます。規格統一された標準ライブラリの他にも様々なライブラリがあります。今回はゲーム作成に使う機能を簡単に利用できる関数にまとめた山田巧氏製作の DX ライブラリを利用して C 言語でプログラムを書きます。

10.1.2 シューティングゲーム

さて、まずは STG というジャンルについて考えましょう。シューティング、というからには自機を操作して飛び道具で敵機を攻撃するのだとおもいます。STG が世に出た当初は二次元的視点のいわゆる 2DSTG でした。固定画面のものから縦スクロール、横スクロール、果ては多方向スクロールなるものまであるとか。その後 3DSTG ができました。読んで字のとおり３次元の空間移動をするゲームです。具体的には宇宙船とか、航空機が登場するジャンルです。それとガンシューティング。これは FPS（First Person Shooting）といい主観視点で空間を移動し標的をうつ方式です 。

シューテイングとひとくくりにいってもたくさんの種類があることがわかりました。まあ、基本

的にはプレイヤーが操作するキャラクターが敵の攻撃を避け、敵を撃墜するものでしょう。

10.2 本題

10.2.1 今回つくるもの

より一般的だと判断した 2D 縦スクロールを採用します。横でもいいですが、より世に出回っていると思ったので（調べたわけではない）。画面を XY の二次元直交座標で表し、プレイヤーがキー入力で操作する自機とそれに対向して機動する敵機を描画します。グラフィックや音楽は素材サイトで配布されているファイルを使わせていただきます。

10.2.2 弾丸

シューティングといえばやはり弾でしょう。思いついた限りでは、

1. 直進弾
2. 多方向弾
3. 相手にむかう弾
4. 誘導弾

とこれだけです。

弾を発射するには例によって関数が必要です。自機は GetJoypadInputState 関数などでキー入力を取得し、武器の弾数を適当な変数にしておき for ループを使って弾数分撃つ。

敵機の場合は発射がなんかやりにくそうです。自機は動きまわるのですし。現在時刻はわかるのですから適当な時間に発射か？とか困りましたが、敵機と自機の距離が一定以内という条件にします。敵機に弾切れはいりませんから次弾との間隔はまったく一定でいいでしょう。発射時の時刻と取得し、現在時刻と一定の差が出来たら発射となります。

障害物に当たった場合、それと飛んで行った弾の座標が画面外に出た場合はチェックして消滅させねばなりません。衝突判定がかなり大変です。

１の直進弾は簡単です。y に少しづつ補正値を加えればいいだけ。さて次、２番です。

方向弾は角度を決めて撃つ弾です。円を描くように撃ったり、前方にばらまくように撃ったりするわけです。時間差つければ螺旋っぽくなるでしょう。そこまでしませんが。これは cos と sin をつかいます。補正値に cos,sin をかけるだけ。これも簡単です。

３は発射時の相手の方向に直進する弾という意味です。誘導弾とは違い、発射後に補正はかかりません。これを撃つには角度を取得せねばなりません。敵機と自機の座標を結ぶベクトルを関数の返り値をとって atan2 をつかえばこれもいけそうですね。atan2 を使うのをググるまで思いつかず割と悩みましたが。

４はそのまんま弾道を修正しつつ相手に向かう弾です。フレーム毎に現在の移動ベクトルと標的へのベクトルの和を新しい移動ベクトルとして計算します。なんか処理が重くなりそうですね。

１から３の弾で十分かと判断します。よって保留。それとボス用の巨大弾も用意したいです。極太レーザーでも可。

10.2.3 敵

敵を出現させます。敵の種類ひとつにつきひとつのヘッダファイルを用意して、その中に処理を書いていくようにします。複数のデータをあつかうので構造体に放り込みましょう。敵の flag をたてて出現させ、画面外にでる、撃墜されるなどのイベントが起これば敵の flag を偽にします。敵の構造体にいれるのは座標のほかに耐久（当たり判定何度かで消える）、それに大きさでしょうか。当たり判定域は画像分のドットサイズよりやや小さくした円形とし、他と重なれば衝突です。ユニットの中心のｘ、ｙ座標をとって三平方の定理をつかい、距離＜半径が真であればってことです。判定処理には新しく Collision 関数を用意します。敵は自機とちがって操作できないので、決められたとおりに移動します。これが難しい。Error など嫌いです、本当に。

軌道

まずは直線的な移動です。敵機の flag をたてて、if 文でカウントが 0 でなければ座標に補正を加え、時間切れで flag を偽にすることです。もうちょっと見てみましょう。二点の座標の直線距離ををとり、それの X．Y 方向の割合にフレームごとの移動量をかけることで X,Y 方向の移動量を決定します。最初に Update し、移動のたびに移動前の画像を消して移動後の画像を Draw します。要はベクトルの成分を足し引きするだけです。

今度は曲線移動です。この場合簡単のため二次曲線のみとします。つまり、半円とか楕円などです。直線移動と同じくｘ、ｙの補正値ｄｘ、ｄｙを計算し、ｙ＝ｙ＋ｄｙ、ｘ＝ｘ＋ｄｘという式を書けばいいでしょう。円軌道を通るには三角関数をつかえばいいでしょう。正弦と余弦は大変役立ちます。現在地と適当な円の中心との角度がわかれば円の接線に沿う運動ができます。弧を描くような動きを想定しています。画面端から一列で複数機現れ自機に覆いかぶさるようなかんじです。放物線軌道は公式から $2aX^2 - bY^2 = 1$ で表されます。円運動と似たようなものです。

10.2.4 今後について

ゲームをつくろうということでやってみましたが、簡素なシューティングだけでも頭が煮えそうです。骨組みをつくって動くようにしても、独自性なんて出せませんし、手直しするととたんに動かなくなったりします。我ながらセンスがない … 。発表するときに完成したといえるか不安です。来年はもっとしっかり作りたいと思います。

10.3 終

ひどく雑な出来の文になってしまいましたが、C の初歩の知識だけでもプログラムを組んで楽しめるということがつたわれば幸いです。ここまでお付き合いいただきありがとうございました。

参考文献

[1] 大槻 有一郎
『14 歳からはじめる C 言語わくわくゲームプログラミング教室』 株式会社ラトルズ

[2] 山田 巧
「DX ライブラリ置き場」
<http://homepage2.nifty.com/natupaji/DxLib/>

[3] 「C 言語入門」
<http://www5c.biglobe.ne.jp/ ecb/c/c00.html>

[4] すきやき
「ゲームプログラミング Wiki」
<http://www.c3.club.kyutech.ac.jp/gamewiki/index.php?FrontPage>

11 軍艦ゲームを作ってみた

情報工学課程 1 回生 杉本 洋介

11.1 始めに

きっかけは、合宿 3 日前に合宿で何かゲームを作ろうと思ったものの題材が思いつかなくて困り果てていた時、何でもいいから案が欲しくて wikipedia で「ボードゲーム」と検索して、「海戦ゲーム」、というものを発見したことでした。そこで、そのゲームを簡略化した「軍艦ゲーム」を作ろうと決めました。合宿ではグラフィックライブラリを使用する予定だったのですが、行き詰ったまま合宿が終わってしまい、とにかくキャラクターベースのゲームとして動作するようにしたいと考えました。

11.2 軍艦ゲームとは

「軍艦ゲーム」とだけ言ってどれだけの人に中身を理解してもらえるか分からないので、簡単なルールに説明をしましょう。まずはお互いに盤の中にいくつかの船を隠します。(盤の大きさ、船の大きさ、船の数などは様々です。)そして、その後交互に相手の盤の中から 1 マスずつを選んで隠れている軍艦を探し出します。(軍艦ゲームという名前ですからここは大砲で相手の船を狙うという表現を使いました。)最終的には先に相手の軍艦のマス全てを当てることができれば勝ちとなります。

11.3 プログラム

使用したプログラミング言語は C 言語です。前述したように、当初はグラフィックライブラリを用いて作ろうと考えましたが、行き詰ってしまったため、今回は用いずに制作しました。

11.3.1 基本骨子

このプログラムの柱になるのは、○×ゲームや五目並べと同様に 2 次元配列です。これを用いて座標を表現しています。

次のソースは制作したプログラムの冒頭部分です。

```
#include <stdio.h>
#define SENKAN 100
#define MISS -200
#define HIT -100

int senkoboard[10][10];
int koukoboard[10][10];
int turn = 0;
int hidden;               //0 で艦を隠し、1 で艦を見せる//
int fin = 0;

int main(void)
{
  printf("先攻後攻を決めてください\n");
  boatset(0);
  boatset(1);
  do{
    shot(turn);
    check(turn);
    turn++;
  }while(fin == 0);
  return 0;
}
```

可能な限り動作のプログラムを main 関数から外に出したので main 関数はこれだけです。これから各関数の内容を簡略に説明します。

11.3.2 boardprint(int turn,int hidden)

この関数の目的は盤面の様子を画面に表示することです。
このゲームでは senkoboard[10][10],koukoboard[10][10] という int 型の 2 つの 2 次元配列を用意します。

```
void boardprint(int turn, int hidden)
{
  int i,j,k;
  switch(turn%2)                //先攻か後攻か//
  {
    case 0:                   //先攻//
      printf("YX");
      for(k=0; k<10; k++){
        printf("%2d",k);
      }
      printf("\n");
      for(i=0; i<10; i++){
```

```
        printf("%2d",i);
        for(j=0; j<10; j++){
          switch(hidden)       //軍艦の位置を見せるかどうか//
          {
            case 0:            //見せない場合//
              if(senkoboard[i][j] == 0 || senkoboard[i][j] == SENKAN)
                printf("〜");
              break;
            case 1:            //見せる場合//
              if(senkoboard[i][j] == 0)
                printf("〜");
              if(senkoboard[i][j] == SENKAN)
                printf("□");
              break;
          }
          if(senkoboard[i][j] == HIT)
            printf("■");
          else if(senkoboard[i][j] == MISS)
            printf("×");
        }
        printf("\n");
      }
      printf("\n");
      break;
    case 1:                    //後攻//
      printf("YX");
      for(k=0; k<10; k++){
        printf("%2d",k);
      }
      printf("\n");
      for(i=0; i<10; i++){
        printf("%2d",i);
        for(j=0; j<10; j++){
          switch(hidden)
          {
            case 0:
              if(koukoboard[i][j] == 0 || koukoboard[i][j] == SENKAN)
                printf("〜");
              break;
            case 1:
              if(koukoboard[i][j] == 0)
                printf("〜");
              if(koukoboard[i][j] == SENKAN)
                printf("□");
              break;
          }
          if(koukoboard[i][j] == HIT)
            printf("■");
```

```
          else if(koukoboard[i][j] == MISS)
          printf("×");
        }
        printf("\n");
      }
      printf("\n");
      break;
  }
}
```

この関数では senkoboard[10][10],koukoboard[10][10] という int 型の 2 つの 2 次元配列を用いて、10*10 の盤面を 2 つ用意して表現しています。また、プログラムの冒頭部分でマクロで SENKAN = 100,HIT = -100 MISS = -200 としています。関数の実引数には turn と,hidden の 2 つがあります。turn は奇数か偶数かによって先攻後攻の分類をしています。hidden は 0 の場合軍艦の位置がわからないように表示して、1 の場合には軍艦の位置をすべて表示します。

11.3.3 boatset(int turn)

この関数の目的は、自分の軍艦を盤面に配置することです。
このプログラムでは、長さ 3 マスの艦を 3 つ、5 マスの艦を 2 つを配置します。まず縦向きにか横向きかを選択して、その後配置したい中心のマスを入力します。そして、入力したマスと、縦向きの場合その上下、横向きの場合左右のマスに対応した配列に 100 を代入します。この関数の間は、自分の配置した場所が見られるように前述の boardprint(int turn, int hidden) への引数は hidden = 1 となります。図 11.1 は、4 つの艦を配置して 5 つ目の配置をしようとしている時の画面です。先攻、後攻ともに配置が完了したらようやく準備完了です。

図 11.1: boatset() の様子

11.3.4 shot(int turn)

この関数の目的は、相手の盤面の中に隠された軍艦を予想して狙いたい座標を入力することです。この関数では、先攻の場合、後攻が配置した盤（配置した場所は見えないようになっています。）から 1 マス選択して、そこに対応した配列の要素が HIT なら命中したという扱いとなり、命中したことを知らせる文を表示させます。代入された数字が HIT でなければ外したということになり、-200 を新たに代入します。

11.3.5 check(int turn)

この関数の目的は、プレイヤーが勝利条件を満たしているかどうか確認することです。
まず先攻の shot() が終了後、後攻の盤を構成する配列の中の HIT の数を数えます。このゲームでは全部で 19 マスの軍艦がありますから、HIT が 19 個あれば全て命中させたということになり、勝利となります。前述した main 関数からも分かるように、
先攻の shot() →先攻の check() →後攻の shot() →後攻の check()
が 1 つのループになっていて、どちらかが勝利の条件を満たせば、ゲーム終了となります。

11.4 これからの目標と課題

このゲームのプログラムの問題は

1. ゲームとして動くことだけを当面の目標に作ったため、非常に無駄の多いプログラムとなっている点
2. 隠された相手の情報を探して 2 人対戦するゲームであるにもかかわらず、1 画面で対戦しなければならず、相手の情報を見ようと思えば簡単に見られてしまう点特に 2 番目の問題は相手の情報が見えてしまえば、このゲームの楽しみは 0 になってしまうのでかなり重要な問題と言えます。

今後の目標としては

1. 現在のプログラムを無駄を少しでもカットすること
2. グラフィックライブラリを用いて改良を施すこと
3. 2 つの画面を用いた対戦ゲームとすること

特に、最後の目標は今の自分の知識ではどういった手法をとればいいのか全く分からないのですが、時間をかけて実現したいと思っています。

11.5 最後に

このゲームを作り始めた当初は「五目並べと同じように 2 次元配列を使えばすぐに完成するはず」なんて安直に考えていたんですが、ずいぶんと時間がかかってしまいました。こんなものでも一応 C 言語の文法書以外は参考図書なしに自分で構想を練って作ったものなので小さな満足感はあります。これもいい勉強でした。この稚拙な文章でゲームのルールやプログラムがちゃんと伝えられているのか不安ではありますが、最後まで読んでいただきありがとうございました。

参考文献

[1] B.W. カーニハン/D.M. リッチー 著 / 石田 晴久 訳
『プログラミング言語 C』 第 2 版 共立出版

12 ライブラリとはなんぞや？

情報工学課程 1 回生 竹中 良

12.1 はじめに

この度、松ヶ崎祭の展示用のシューティングゲームを製作するにあたって、さまざまなツールを使いました。その中に、ライブラリがあります。私が今回使ったライブラリは、部活の先輩が作成したものでした。

なので、そのときライブラリについて自分なりに調べてみたことを紹介します。

12.2 そもそもライブラリとは

ライブラリ (Library) は、「図書館」という意味ですが、プログラミングの世界で言えば、複数の関数や変数等の機能をまとめたファイルのことを指します。ライブラリは、Windows 環境では一般的に「.lib」という拡張子を持ちます。ちなみに、ライブラリの定義は次のようになっています。

ある特定の機能を持ったプログラムを、他のプログラムから利用できるように部品化し、複数のプログラム部品を一つのファイルにまとめたもの。

つまり、何かプログラムを作成する際に用いることのできるツールのようなものです。なので、ライブラリ自体は単独で実行することはできず、（実行ファイルではない）他のプログラムの一部として動作するものです。

12.3 ライブラリのいろいろ

C 言語の普及に伴ってライブラリが多数生まれましたが、1989 年に ANSI による C 言語の標準規格が制定されることで統一化が図られ、更にいくつかの新たな概念が導入され、それが標準 C ライブラリとなりました。標準Cライブラリには, 入出力機能を定義してある stdio.h や文字列操作の機能を定義してある string.h、数学的な演算を行うための関数を定義してある math.h など、様々な標準ヘッダがあります。

ちなみに、C言語のプログラムを勉強し始めたころに使う printf 等の関数は、標準ライブラリの stdio.h を用いることで使えるのです。

また、ソフトウェアの開発メーカーなどが市販したり、あるいは宣伝もかねて無償で提供したりといった場合も多いので、さまざまな種類があります。

12.4 ライブラリの使い方

私はまだC言語しか勉強しておらず、他の言語は知らないのでここではC言語について紹介します。使い方といっても、そのライブラリのファイルをダウンロードし、ヘッダファイルをプリプロセッサの#include で読み込むだけです。つまり、

#include ”ヘッダファイル名”
または
#include <ヘッダファイル名>

のようにすれば、ヘッダで定義してある機能を使うことができます。「<>」で囲む場合と、「””」で囲む場合では意味が異なります。「<>」で囲む場合は, コンパイラが用意してくれるヘッダファイルです。例えば標準関数の宣言をしているヘッダーファイルを読み込む場合などで、コンパイラが読み込むファイルの場所を自動的に探してくれます。

これにより、定義の手間が省け、楽にプログラミングできるようになります。
また、コンパイルを行うと、ソースファイルを元に、オブジェクトファイルと呼ばれるファイル (.obj) が生成されます。これは、ソースファイル中に記述した関数の中身が、実際に実行できるような状態に変換された結果が保存されています。しかし、この状態では、それぞれのファイルがばらばらになってしまっており、１つの実行可能ファイル (.exe) になっていませんから、実行できません。これらから実行可能な.exe ファイルを作成する過程が「リンク」です。ですから、単に「コンパイル」しただけでは、実行ファイルは完成しません。

.lib ファイルは、.obj の情報をまとめた状態です。依然として、それだけでは実行できませんが、その内容をリンクすれば、実行可能なファイルを生成できます。.lib の目的は、よく使う機能群を１つにまとめておくことにあります。そうすることで、同じ機能を、別のプログラムでもリンクさえすれば使えるようになる訳です。
また、ライブラリには静的ライブラリと共有ライブラリと動的ライブラリと、いろいろ種類があります。
静的ライブラリは、通常のオブジェクトファイルの単なる集合体です。これは、実行可能プログラムを作成するときにコンパイルとリンク処理の一部として取り込むことで使用できます。
共有ライブラリは、プログラム起動時にロードされるライブラリです。共有ライブラリが適切にインストールされると、その後に起動される全てのプログラムは、自動的にその新しい共有ライブラリを使うことになります。
動的ライブラリは、標準的なオブジェクトファイルや共有ライブラリとしてビルドされています。主な違いは、プログラムのリンク時や起動時に自動的にはロードされない、という点です。

12.5 ライブラリを作ってみた

静的ライブラリを作ってみました。その内容は、ただ「Hello!」と表示するだけです。

```
//test.c
#include<stdio.h>
#include"test.h"
void hello(){
  printf("Hello!");
}

//test.h
void hello();

//zikko.c
#include"test.h"
int main(){
  hello();
}
```

とりあえずこの３つのファイルを作成します。そして、まずオブジェクトファイル（.o または.obj）を作ります。まず、コンパイラでその２つのファイルが存在するファイルをカレントディレクトリにしておきます。そして、

```
  gcc -o test.c -o test.o
```

とすると、拡張子.o のオブジェクトファイルができます。さらに、

```
  ar rcs libtest.a test.o
```

とし、libtest.a ファイルをつくります。これが静的ライブラリファイル (.a) です。そして、zikko.c をコンパイルします。

```
  gcc -o zikko.exe zikko.c -L. -ltest
```

-l オプションで実行ファイルにリンクさせています。そうしてできた実行ファイル (.exe) を実行すれば

```
  Hello!
```

と表示されます。

12.6 おわり/これから

まだプログラムを始めたばかりのころ、どのサンプルソースを見ても、#include<stdio.h>とあり、それがどういった役割なのかずっと気になっていました。今回学祭用にシューティングゲームを作っているのですが、そのときに調べてみて、やっと納得がいきました。これで、あとはゲームの完成に向けて頑張りたいと思います。

参考文献

[1] 「Wikipedia ライブラリ」
<http://ja.wikipedia.org/wiki/
%E3%83%A9%E3%82%A4%E3%83%96%E3%83%A9%E3%83%AA>

[2] 「C 言語入門」
<http://www5c.biglobe.ne.jp/~ecb/c/c00.html>

[3] K.Y
「Programing Place」
<http://www.geocities.jp/ky_webid/win32c/056.html>

[4] David A. Wheeler 著 / 川崎 貴彦 訳
「Program Library HOWTO」
<http://www.linux.or.jp/JF/JFdocs/Program-Library-HOWTO/index.html>

編集後記

皆様はじめまして、編集担当の中井です。Lime40 号はいかがでしたでしょうか。今回の Lime の編集の印象というと、集まりが悪かったです。特に 3 回生の記事の集まりが悪かったです。これを書いている私も記事の提出が遅れてしまいました。今年の 1 回生は早々に提出してくれたのになんだか申し訳ない気分です。

今回はシューティングゲームに関する記事が 3 つもありますね。作りやすいのでしょうね。まあ、そういう私もシューティングゲームについて書いた訳ですが。

まあ、何にしても無事今回も発行することが出来ましたので良しとしましょう。次号の Lime にも是非ご期待ください。では、では。

平成 21 年 11 月 19 日 編集担当 中井 道

Lime Vol.40

平成 21 年 11 月 21 日 発行 第 1 刷

発行 京都工芸繊維大学コンピュータ部
http://www.kitcc.org/