

令和 7 年 1 月 5 日
京都工芸繊維大学コンピュータ部

Lime 64

はじめに

コンピュータ部部長の石倉です。この度はお忙しい中、我々コンピュータ部にご興味、ご関心を持っていただきありがとうございます。そして、この Lime64 号を見にきてくださったことを、部を代表して厚く御礼申し上げます。この Lime は部員の活動の一部を記したものです。日々の活動の成果や部員個人の趣味など、内容は多岐にわたります。これを通じて、コンピュータ部のことを知っていただき、少しでも我々の活動に興味を持っていただければ幸いです。そして、今回の Lime 作成にあたり編集を担当した樋口さん、記事を執筆した部員各位、ならびに OB・OG の方々を含め、日々我々の活動を支えてくださっているすべての方々への感謝をもって始めの挨拶とさせていただきます。

令和 6 年 11 月 24 日

京都工芸繊維大学コンピュータ部部長 石倉 朋佳

目次

はじめに	iii
1 OS 自作の本を読んで途中で実装 — 石倉 朋佳	1
2 vim 入門 — 近藤 偉希	7
3 数当てゲーム — 樋口舞子	13
4 C# 言語による一般的なすごろくの開発 — 田中大貴	19
編集後記	32

1 OS 自作の本を読んで途中まで実装

情報工学課程 3 回生 石倉 朋佳

1.1 はじめに

1 回生のときに OS 自作という分野があることを知り、やってみたいと思っていた. なかなか手を出せずにいましたが, この夏こそやってみよう, やってみたいと思い, OS 自作に挑戦することにしました.

今回読んだ「作って理解する OS x86 系コンピュータを動かす理論と実装」[1] では, x86 系の CPU をもつ PC のオペレーティングシステム (OS) を対象として, OS の入門の説明がされています. OS の入門の説明は, コンピュータの基礎的な説明から始まっており, 本の多くのページを使って説明されています.

有名な OS 自作の本の一つに C++ で書く蜜柑が描かれた表紙の本 [2] がありますが, アセンブリ言語で書きたかったので「作って理解する OS」を読んで実装しました. これは, 3 回生前期の授業で CPU について学び, アセンブリ言語で簡単なプログラムを実行する体験をして, アセンブリ言語のレベルでプログラムを考えるのが面白かったからです.

本記事ではどのようなものを実装していたか, どのように実装していたかについて説明します. オリジナルな機能を実装する予定でしたが, その段階まで至らなかったため, 実装した内容はほとんど本 [1] の通りです.

1.2 開発環境と開発の流れ

x86 系 CPU を搭載したパソコンを持っていないため, 今回は実機での動作確認はできません. 動作確認には, QEMU[3] を用いました. QEMU は本で紹介されているエミュレータの一つで, CPU や周辺機器なども含めたシステム環境の動作を確認できます. ソースコードはアセンブリ言語で書き, アセンブラは NASM[4] を用いました. また, ソースコードを書くためのエディタとして VSCode やメモ帳を, アセンブルやプログラムの起動のために Windows PowerShell を用いました. 環境構築の方法は本に書かれている通りなので省略します.

開発の流れは以下に示す通りです. プログラムの実装, アセンブル, エミュレータで実

行して確認，を繰り返し，機能を少しずつ追加していききました。

1. アセンブリ言語でソースファイル (.s) を書く
 - (a) 新たにソースファイルを作成し，新たな関数を実装する
 - (b) カーネルのソースファイル kernel.s に変更を加えて，新たに実装した関数を使う
2. アセンブルする
3. 生成された実行イメージ (.img) をエミュレータを用いて確認する
4. 1 に戻って新たに機能を追加する

OS に追加する機能はモジュールに分けて，基本的に関数の形で実装しました。「作って理解する OS」では，アセンブラのマクロを用いることで，アセンブリ言語で書いた関数を呼び出して使えるようにしています。マクロ `cdecl` は C 言語の関数呼び出しと同様の処理を行うように定義されたマクロであり，カーネルのソースファイル `kernel.s` や関数のソースコードの中で関数を呼び出すときに用います。

1.3 実装した OS の画面

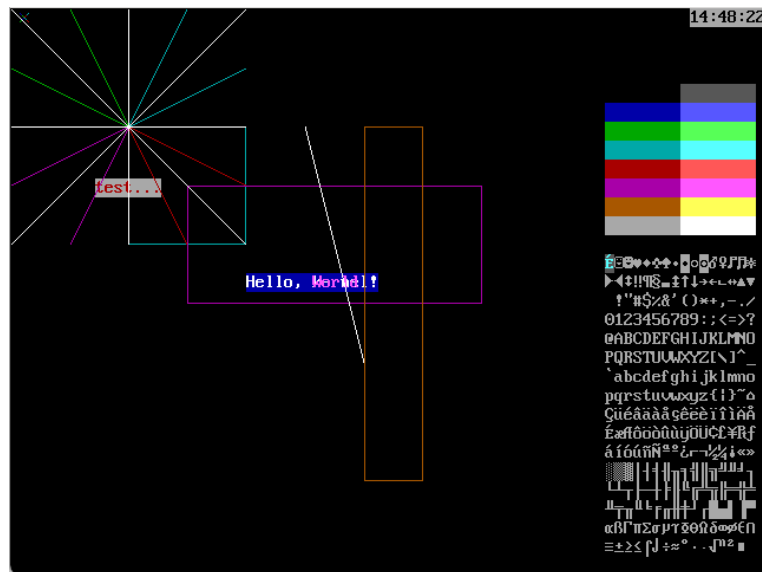


図 1.1 実装途中の OS をエミュレータで実行したときの画面

図 1.1 は実装途中の OS をエミュレータで実行したときの画面です。まだ OS っぽいことはできず，画面への線や文字の表示，現在時刻の表示だけができます。画像では分かりませんが，現在時刻はリアルタイムで更新されています。

1.4 関数の実装

実装した関数の例として、長方形を描画する関数 `draw_rect` をソースコード 1.1 とソースコード 1.2 に示します。 `draw_rect` 関数は、引数に渡された始点と終点に従って直方体を画面に描画する関数です。 戻り値は無く、第 1 引数に始点の X 座標、第 2 引数に始点の Y 座標、第 3 引数に終点の X 座標、第 4 引数に終点の Y 座標、第 5 引数に描画色を渡します。

```
1      draw_rect:
2          ;      【スタックフレームの構築】
3          push    ebp                ;      + 4 | IP   (戻り番地)
4          mov     ebp, esp          ;      EBP+ 0 | BP   (元の値)
5
6          ;      【レジスタの保存】
7          push    eax
8          push    ebx
9          push    ecx
10         push    edx
11         push    esi
12
13         ;      【処理の開始】ここに主な処理を書く
14         ;      //   ( draw_rect() ではソースコード 2 の内容が入る)
15
16         ;      【レジスタの復帰】
17         pop     esi
18         pop     edx
19         pop     ecx
20         pop     ebx
21         pop     eax
22
23         ;      【スタックフレームの破棄】
24         mov     esp, ebp
25         pop     ebp
26
27         ret
```

ソースコード 1.1 主な処理以外の部分 (`draw_rect` 関数の場合)

今回作成した関数は、いずれも以下のような構成で実装しました。このうち、主な処理の部分はソースコード 1.2 に、それ以外の部分がソースコード 1.1 にあたります。

- スタックフレームの構築
- レジスタの保存
- 主な処理 (引数を汎用レジスタへコピー、目的の処理)

- レジスタの復帰
- スタックフレームの復帰

関数を呼び出す前後で、レジスタの状態は変わらないようにします。関数の目的の処理をするためにレジスタを使うため、関数を呼び出す前の値もスタックに退避させる必要があります。関数によって目的の処理でどのレジスタを使うかは変わるため、どのレジスタを保存させるかは関数によって変える必要があります。また、関数の主な処理が終わった後、スタックに退避させていたレジスタの値を戻す必要があります。退避させるときに push した順番と逆順に pop することで、各レジスタは関数を呼び出す前の状態に戻ります。

スタックフレームの保存と破棄の部分のコードは基本的にどの関数でも同じです。スタックのベースポインタ（ここでは EBP）に加算すると戻り番地が得られ、さらに加算すると順に引数を得ることができます。関数 draw_rect では用いていませんが、ベースポインタから減算して局所変数を確保することもできます。関数 draw_rect のスタックフレームは以下のようになっています。

```
; -----|-----  
;    +24| 引数 5 (color) 色  
;    +20| 引数 4 (Y1) 終点  
;    +16| 引数 3 (X1) 終点  
;    +12| 引数 2 (Y0) 始点  
;    + 8| 引数 1 (X0) 始点  
;    + 4| IP (戻り番地)  
; EBP+ 0| BP (元の値)  
; -----|-----
```

```

1      ;    【処理の開始】
2      mov     eax, [ebp + 8]          ; EAX = X0;
3      mov     ebx, [ebp +12]         ; EAX = Y0;
4      mov     ecx, [ebp +16]         ; EBX = X1;
5      mov     edx, [ebp +20]         ; EAX = Y1;
6      mov     esi, [ebp +24]         ; EAX = 色;
7
8      ;    座標の大小関係を確定
9      cmp     eax, ecx                ; if (X0 >= X1)
10     jl      .10E                    ; {
11     xchg    eax, ecx                ;   X0 と X1 を入れ替える
12     .10E:                          ; }
13
14     cmp     ebx, edx                ; if (Y0 >= Y1)
15     jl      .20E                    ; {
16     xchg    ebx, edx                ;   Y0 と Y1 を入れ替える
17     .20E:                          ; }
18
19     ;    矩形を描画
20     cdecl   draw_line, eax, ebx, ecx, ebx, esi ; 上線
21     cdecl   draw_line, eax, ebx, eax, edx, esi ; 左線
22
23     dec     edx                      ; EDX--; // 下線はドット下げる1
24     cdecl   draw_line, eax, edx, ecx, edx, esi ; 下線
25     inc     edx
26
27     dec     ecx                      ; ECX--; // 右線はドット左に移動1
28     cdecl   draw_line, ecx, ebx, ecx, edx, esi ; 右線

```

ソースコード 1.2 draw_rect 関数の主な処理 ([1]pp.516-517 の注釈に変更を加えたもの)

はじめに、2 から 6 行目では mov 命令を用いて引数に渡された値をレジスタに格納しています。

また、9 から 12 行目は、コメントに示す通り、C 言語では if 文にあたる処理をしています。cmp eax, ecx では、eax の値から ecx の値を引いてその結果をフラグに格納し、2 つのオペランドの値を比較します。レジスタ eax と ecx にはそれぞれ第 1 引数と第 3 引数の値が格納されたので、つまり関数に渡された始点の X 座標と終点の X 座標の値を比較しています。

jl 命令は Jump is less の略で、「より小さい」条件を満たすときオペランドのラベルにジャンプします。ここでは、直前が cmp 命令なので eax(X0) が ecx(X1) より小さいとき、ラベル.10E にジャンプして以降の処理を実行します。そうでないとき、そのまま次の行の命令を実行するため、ここでは xchg 命令を用いて eax(X0) と ecx(X1) の値を交換します。

`draw_line` は、実装済みの関数で、引数に渡された始点と終点、描画色に従って画面に直線を描画します。 `draw_rect` 関数と同様に、第 1 引数に始点の X 座標、第 2 引数に始点の Y 座標、第 3 引数に終点の X 座標、第 4 引数に終点の Y 座標、第 5 引数に描画色を渡します。マクロ `cdecl` によって `draw_line` 関数を呼び出し、長方形の各辺を描画していくことで目的の長方形を描画します。

1.5 おわりに

今回、肝心のマルチタスクの実装や後半のファイルシステムの実装などは、期間内に終わらずできませんでした。本当はオリジナルの機能を作りたいと思っていましたが、思わぬところでうまくいかず進まなくなったり、コードの意味を理解するのに時間がかかったりするなど、思っていた以上に進まなかったです。夏の間に終わることはできませんでした。まずは実装していない部分を実装して、オリジナルな部分も実装できるようにしたいと思います。

まだ本も終わられていませんが、前からやってみたいと思っていたことの一つに着手できたのは良かったです。本を読んだり実装したりするときに、2 回生・3 回生のこれまでの授業で学んだことやしたこととの関連を色々なところで感じました。やり始めるだけなら 1 回生のときにもできたかなと思うので、やってみたいと思ったことは後回しにせず、まずはもっと試してみよう、と思います。

参考文献

- [1] 林高勲「作って理解する OS x86 系コンピュータを動かす理論と実装」技術評論社、2019 年 10 月
- [2] 内田公太「ゼロからの OS 自作入門」マイナビ出版、2021 年 3 月
- [3] QEMU, 最終閲覧日 2024 年 11 月 23 日, <https://qemu.weilnetz.de/w32/>
- [4] NASM, 最終閲覧日 2024 年 11 月 23 日, <https://www.nasm.us/pub/nasm/>

2 vim入門

電子システム工学過程 2 回生 近藤 偉希

2.1 はじめに

私が vim の存在を知ったのは、OB さんの紹介がきっかけでした。「効率的に編集できる」と勧められ、興味を持ちました。使い始めて半年程度なので、まだ初心者ですが、学んだことをこの文章にまとめることで、読者の皆さんにも vim の魅力を伝えられたらと思います。vim の基本的な使い方や主要なコマンドについて紹介します。どうぞよろしくお願いいたします。

2.2 そもそも vim とは

vim は、テキストエディタの 1 つです。特徴として、キーボードだけで操作が完結するため、高速で効率的に編集が行えます。現在、多くの人が vscode を使って満足しているかもしれません。ですが、vim はマウスを使わずに編集を完結できることから、プログラミングにおいて非常に便利です。また、vscode にも vim の機能を使えるようにする拡張機能があるため、興味があれば手軽に試すことができます。vim は多くのコマンドがあり、使いこなすのに時間がかかることというデメリットもあります。しかし、一度慣れるとホームポジションから手を離すことなく操作できるため、編集速度が向上します。

2.3 モード

vim には 4 つの主なモードがあります。それぞれのモードは、以下のような用途に適しています。このように、vim ではモードを切り替えることで、効率的な編集作業を実現しています。

- **ノーマルモード**: ファイル内の移動や、コピー・貼り付けなどの編集操作を行うモード。
 - － 例: カーソル移動 (h, j, k, l)、行削除 (dd)、貼り付け (p)。
- **挿入モード**: 文字や文章を直接入力するモード。
 - － 例: 挿入を開始 (i)、行末に追加 (A)。
- **ビジュアルモード**: テキストの範囲選択を行うモード。
 - － 例: 選択した範囲を削除 (d)、コピー (y)。
- **コマンドラインモード**: 保存や置換、検索などのコマンドを実行するモード。
 - － 例: 保存 (:w)、終了 (:q)。

2.4 ノーマルモード

ノーマルモードの主要なコマンドを表 1.1 に示す。C-? はコントロールキーと?を同時に押す。

表 2.1 ノーマルモードの主なコマンド

コマンド	コマンドの内容
j	カーソルを一行下へ移動する
k	カーソルを一行上へ移動する
h	カーソルを一文字左へ移動する
l	カーソルを一文字右へ移動する
gg	ファイルの 1 番上へ移動する
G	ファイルの 1 番下へ移動する
w	次の単語の先頭へ移動する
e	次の単語の末尾へ移動する
b	前の単語の先頭へ移動する
caw	カーソルの位置にある単語を消して挿入モードに入る
\ci{	{ } の中にある文字を消して挿入モードに入る
\ci(	() の中にある文字を消して挿入モードに入る
\ci <	<> の中にある文字を消して挿入モードに入る
i	カーソルの左の位置から挿入モードに入る
I	カーソルのある行の先頭の位置から挿入モードに入る
a	カーソルの右の位置から挿入モードに入る
A	カーソルのある行の末尾の位置から挿入モードに入る
dd	1 行切り取る
yy	1 行コピーする
p	カーソルの下の行に貼り付ける
P	カーソルの上の行に貼り付ける
.	前のコマンドを実行する
u	前のコマンドを実行を取り消す
C-a	カーソルより後ろにある数字を +1 する。
C-x	カーソルより後ろにある数字を -1 する。

このように様々なコマンドがあるが、この中で大切なのは「.」コマンドである。これは、ノーマルモードから挿入モードに入ってノーマルモードに戻るまでの操作を覚えているので、うまく使うと非常に効率的になる。

2.5 挿入モード

挿入モードの主要なコマンドを表 1.2 に示す。C-? はコントロールキーと?を同時に押す。

表 2.2 挿入モードの主なコマンド

コマンド	コマンドの内容
esp キー	ノーマルモードに戻る
C-[	ノーマルモードに戻る
C-h	バックスペース
C-w	1 単語を消す
C-u	カーソルの位置から行の先頭までを消す

ノーマルモードに戻るキーは 2 つあるが、ホームポジションから離れないですむ C-[をおすすめする。C-h のいいところは、ホームポジションからバックスペースを押すよりも素早く楽だからである。また、これは CapsLock キーをコントロールキーに変更していると素早く押すことができるので便利である。windows の場合、powertoys を使うと簡単に変更できる。詳しくは自分で調べてみてほしい。C-h , C-w , C-u は、vim の機能ではないが、非常に便利です。

2.6 ビジュアルモード

ビジュアルモードは、ノーマルモードから v, V を入力することで入ることができます。v だと文字単位で、V だと行単位で動きます。カーソルの動かし方は、ノーマルモードの時と一緒です。行を指定してから d, y などで切り取ったりコピーしたりできます。

2.7 コマンドラインモード

コマンドラインモードは、ノーマルモード中に「:」を入力することで入ることができます。コマンドラインモードの主要なコマンドを表 1.3 に示す。

表 2.3 コマンドラインモードの主なコマンド

コマンド	コマンドの内容
:w	保存する
:wq	保存して終了する
:%s/(置換前)/(置換後)g	置換することができる

2.8 終わりに

ここまでで、vim の基本的な使い方や主要なコマンドについて紹介しました。最初は覚えることが多くて大変に感じるかもしれませんが、慣れてくるとその効率性に驚くはずで
す。vim には紹介しきれなかった便利なコマンドがたくさんあります。ぜひ、自分に合っ
た使い方を見つけてみてください。少しでも vim を始めるきっかけになれば幸いです。
最後までお読みいただき、ありがとうございました。

参考文献

- [1] 実践 vim 思考のスピードで編集しよう 出版社 アスキー・メディアワークス 発売日 2013/8/29

3 数当てゲーム

電子システム工学課程 2 回生 樋口舞子

3.1 はじめに

大学の授業で C 言語について学んだので、C 言語を使った簡単なゲームを作成した。

3.2 数当てゲームの概要

プレイヤーがキーボードから数値を打ち込み、コンピュータが用意した「当てさせる数」の大小を判定する。制限回数以内にプレイヤーが「当てさせる数」を当てられたら勝ちというゲームである。

3.3 作成コード

10 回以内に 0 ~9 の整数を当てる数当てゲームを作成した。入力履歴を表示する使用になっている。

```
1      #include <time.h>
2      #include <stdio.h>
3      #include <stdlib.h>
4
5      #define MAX_STAGE 10
6
7      int main(void){
8
9          srand(time(NULL));
10
11          int ans = rand() % 10;
12          int no;
13
14          int num[MAX_STAGE];
15          int stage = 0;
16
```

```
17     printfの整数を当てよう("0~9!!\n\n");
18
19     do{
20         printf残り("%回。いくつかな:d", MAX_STAGE - stage);
21         scanf("%d", &no);
22         num[stage++] = no;
23
24         if(no > ans)
25             printfもっと小さいよ。("\a\n");
26         else if(no < ans)
27             printfもっと大きいよ。("\a\n");
28     } while (no != ans && stage < MAX_STAGE);
29
30     if(no != ans)
31         printf残念。正解は("%でした。d\a\n",ans);
32     else{
33         printf正解です。("\n");
34         printf("%回で当たりましたね。d\n", stage);
35     }
36
37     puts("\n--- 入力履歴 ---");
38     int x;
39     for (int i = 0; i < stage ; i++){
40         if(num[i]-ans == 0){
41             x = i;
42             continue;
43         }
44         printf(" %2d : %4d %+4d\n", i+1, num[i], num[i]-ans);
45
46     }
47     printf(" %2d : %4d %4d\n", x+1, num[x], 0);
48
49
50     return 0;
51 }
```

ソースコード 3.1 作成コード

3.4 コード解説

3.4.1 rand 関数

rand 関数で乱数を生成できる。srand 関数により乱数生成のための種を設定する。種の値が決まってしまうと、乱数の系列が決まってしまう。乱数の系列をランダムにするた

め、現在の時刻に基づいて乱数を決定するようにする。入力回数に制限を設ける場合の数当てゲームのコードを示す。

```

1      #include <time.h> 時刻//^^ef^^bd^^日付のヘッダa5
2      #include <stdio.h>
3      #include <stdlib.h> 一般ユーティリティのヘッダ//
4      int main(void){
5          srand(time(NULL)); 乱数の種を設定 //
6          const int max_stage = 10; 最大入力回数//
7          int remain = max_stage; 残り何回入力できるか //
8
9          int ans = rand() % 1000 の乱数を生成 //0~999
10         int no; 読み込んだ値//
11
12         printfの整数を当てよう("0~999!!\n\n");
13         do{
14             printf残り("%回。いくつか：d", remain);
15             scanf("%d", &no);
16             remain--;
17
18             if (no > ans)
19                 printfもっと小さいよ。("\a\n");
20             else if (no < ans)
21                 printfもっと大きいよ。("\a\n");
22         }while (no != ans && remain > 0);
23
24         if(no != ans)
25             printf残念。正解は("%でした。d\a\n", ans);
26         else{
27             printf正解です。("\n");
28             printf("%回で当たりましたね。d\n", max_stage - remain);
29         }
30
31         return 0;
32     }
```

ソースコード 3.2 rand 関数

3.4.2 入力履歴の表示

プレイヤーの入力した値を保存することで、当てさせる数にどれだけ近づいたか (または離れたのか) を、ゲーム終了時に確認できるようにする。ソースコード 3.2 に入力履歴を表示する機能を追加したものを示す。

```

1  #include <time.h>
2  #include <stdio.h>
```

```
3  #include <stdlib.h>
4
5  #define MAX_STAGE 10 最大入力回数 //
6
7  int main(void){
8
9      srand(time(NULL)); 乱数の種を決定 //
10
11     int ans = rand() % 1000; の乱数を生成//0~999
12     int no; 読み込んだ値//
13
14     int num[MAX_STAGE]; 読み込んだ値の履歴//
15     int stage = 0; 入力した回数 //
16
17     printfの整数を当てよう("0~999!!\n\n");
18
19     do{
20         printf残り("%回。いくつか：d", MAX_STAGE - stage);
21         scanf("%d", &no);
22         num[stage++] = no; 読み込んだ値を配列に格納 //
23
24         if(no > ans)
25             printfもっと小さいよ。("\a\n");
26         else if(no < ans)
27             printfもっと大きいよ。("\a\n");
28     } while (no != ans && stage < MAX_STAGE);
29
30     if(no != ans)
31         printf残念。正解は("%でした。d\a\n",ans);
32     else{
33         printf正解です。("\n");
34         printf("%回で当たりましたね。d\n", stage);
35     }
36
37     puts("\n--- 入力履歴 ---");
38     for (int i = 0; i < stage ; i++)
39         printf(" %2d : %4d %+4d\n", i+1, num[i], num[i]-ans);
40
41     return 0;
42 }
```

ソースコード 3.3 rand 関数

- 入力された値の配列への格納
格納の様子を図 3.1 に示した。
- 入力履歴の表示

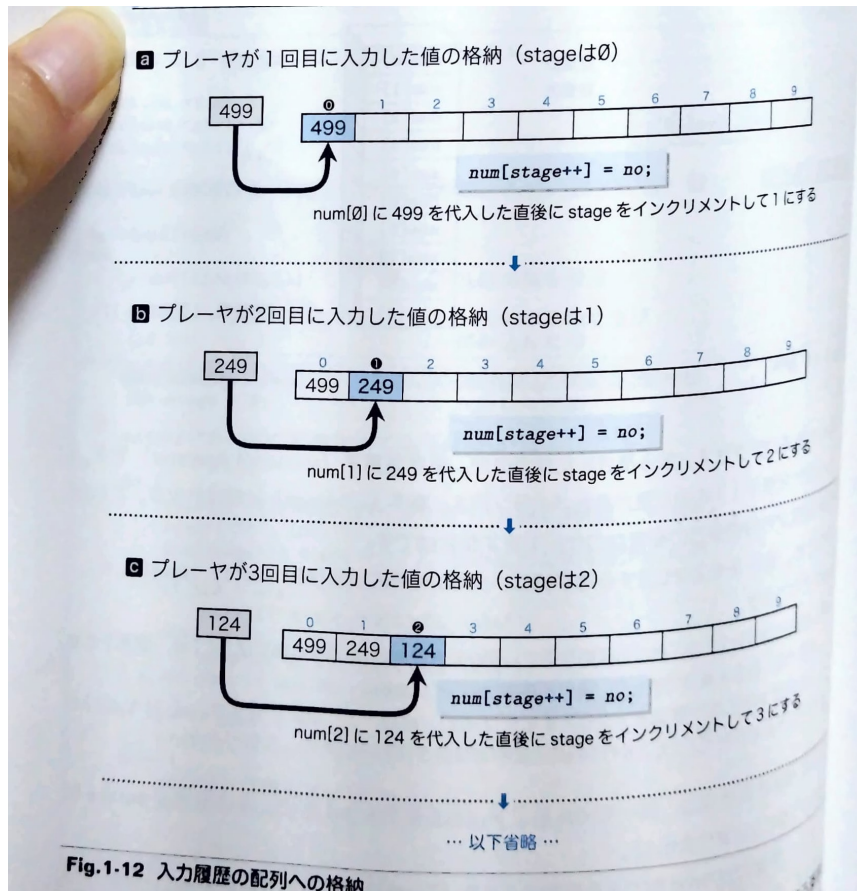


図 3.1 入力履歴の配列への格納

ゲーム終了後は、プレイヤーが入力したすべての値を、for 文を使って表示する。
この様子を図 3.2 に示した。

3.5 さいごに

上記のことから分かる通り、`<stdio.h>`以外のヘッダを使用することで、様々な機能を追加できる。今回は開発にあまり時間を割く事が出来ず、簡単なものしか作成できなっ

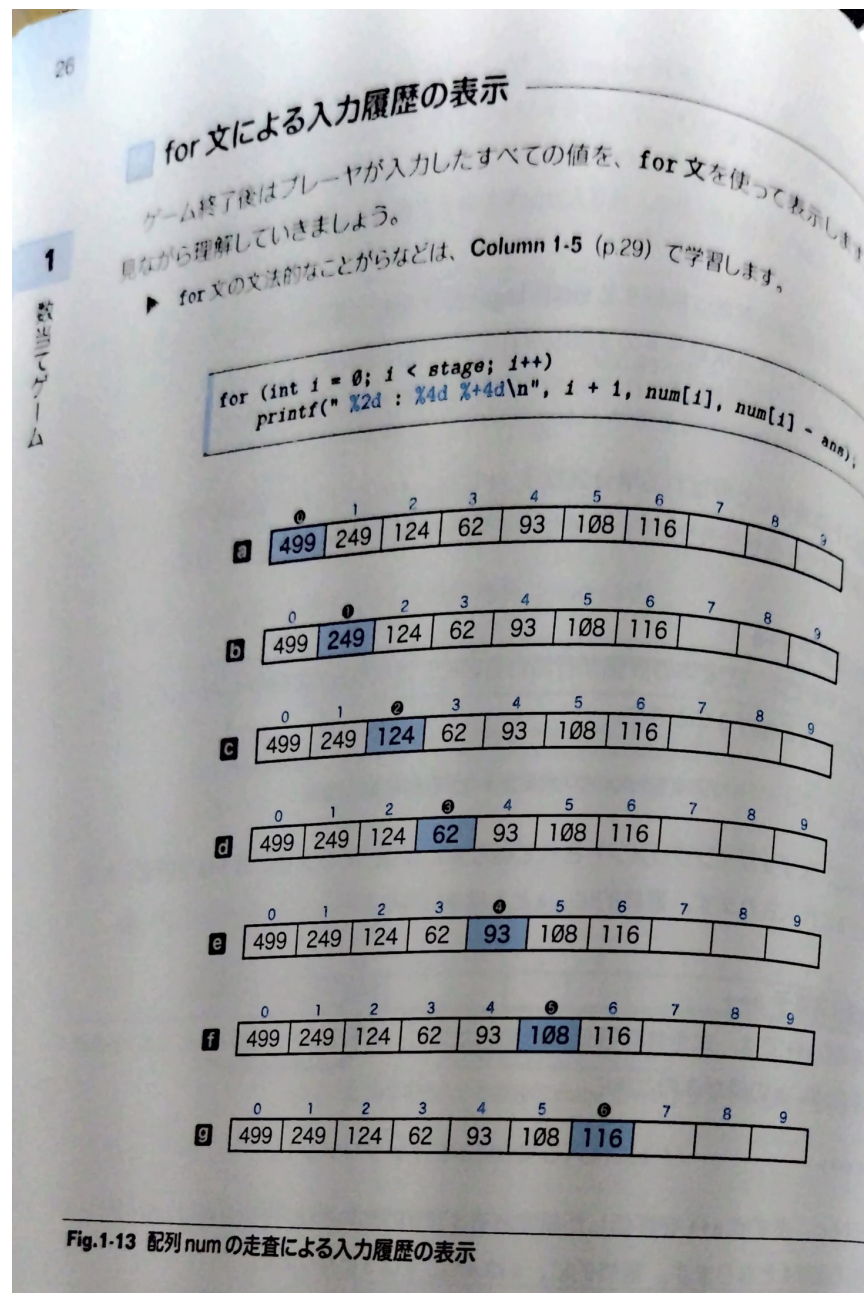


図 3.2 配列 num の走査による入力履歴の表示

た。今後はより難易度の高いゲーム開発に取り組んでいきたい。

参考文献

- [1] 柴田望洋『新明解 C 言語 中級編』第 2 版, SB クリエイティブ, 2022 年

4 C# 言語による一般的なすごろくの開発

情報工学課程 2 回生 田中大貴

4.1 はじめに

私はこれまで C 言語を用いて双六をいくつか作りました。しかし、それらは全て一般的とは言えないものでした。

我々の活動を外部に公開することを考えると、様々な人に受け入れられるものを作るべきであることは自明です。そこで、一般的な双六を作成することとしました。

私はグラフィカルユーザーインターフェース (以下では GUI と表記します。) を作成するために C# を学習しています。しかし、普段はコマンドラインユーザーインターフェース (以下では CLI と表記します。) を用いることを想定した開発を行っているため、慣れないことも多いです。そのため、改善の余地があるかと思いますが、私の成長の記録と思い、修正点を思いつかれても、心の中にそっとしまっておいて頂けると幸いです。

また、私個人でも自身で設定した要求仕様を満たさない部分をいくつか発見し、その部分の修正案を考えました。修正点を挙げる際に、なぜその点を修正するべきかを数式を用いて説明していますので、そのような内容に苦手意識がある方は読み飛ばして頂けると幸いです。

前置きが長くなりましたが、お忙しい中私の記事を読んでいただくことに厚く御礼を申し上げます。

4.2 概要

私は, GUI 開発を行う前に, どのような考え方で開発するかを考えるために初めに C 言語で CLI によるプログラムを書きます. このプログラムで満足できるものを実装できたときに, C# による GUI 開発を始めます. その際に C 言語のコードを C# のコードに書き換える, という手法を用いています. そのため, CLI と GUI の 2 種類の開発を行っています.

4.2.1 目標

製作品について

- 双六を成立させる.
- ゲームバランスを崩壊させないこと

用いる手法について

- 画面遷移を組み込む
- 構造体を理解する
- マップの生成について適切な手法を選択する (配列または連結リスト)
- プレイヤーの順序のシャッフルを行う機能の実装
- 設定を 1 度確定すると変更できなくする機能の実装 (モーダルウィンドウ)

4.2.2 CLI 開発

使用言語 … C 言語

使用したソフトウェア … Visual Studio Code

4.2.3 GUI 開発

使用言語 … C# 言語

使用したソフトウェア … Visual Studio 2022

使用したテンプレート … Windows Form Application

4.2.4 ルール

- 自分の番が回ってきたとき、さいころをシステム上で 1 回投げます。このとき、1 以上 6 以下の自然数が返されるのでその数だけシステム上で駒が進みます。
- マスに止まったとき、マスのイベントを確認します。このすぐろくには以下に示す 6 種類のマスがあります。
 - － **プラスマス** … もう 1 度さいころを投げ、出た目の数だけ先に進みます。
 - － **マイナスマス** … もう 1 度さいころを投げ、出た目の数だけ後ろに戻ります。
 - － **空マス** … 何も起こらず、ターンが終了します。
 - － **休みマス** … 1 回休みとなり、次のターンではさいころを投げられなくなります。
 - － スタートマス … すべてのプレイヤーはこのマスを出発し、後述のゴールへ向かいます。
 - － ゴールマス … このマスに最も早くたどり着いた人が勝ちとなります。
- マスのイベントに応じ、最大 1 回追加でさいころを投げます。
- 追加でさいころを投げて止まったマスのイベントは無効となります。
- ゴールマスは通り過ぎてもゴールしたとみなします。

4.2.5 ゲームの流れ

1. 1 つめの画面で, 2 人以上のプレイヤーの名前, 合計 1 マス以上になるようなマスの内訳, プレイヤーの順序をシャッフルするか否かを入力します.
2. 2 つめの画面で内容を確認し, 問題があれば 1 つめの画面に戻り, 内容を修正します. 問題がなければ次の画面に進むボタンを押します.
3. 3 つめの画面に進むにあたり, 以下の条件をすべて満たしている必要があります.
 - プレイヤーの人数が 2 人以上, 4 人以下であること
 - **プラスマス**, **マイナスマス**, **空マス**, **休みマス** の 4 種類のマスの出現数の合計が 1 以上 4000 以下であり, それぞれの出現数が 0 以上 1000 以下であること
 - プレイヤーの順序をシャッフルするか否かが明示されていること
4. これらすべてを満たす場合のみ 3 つめの画面に進むことができます. 少なくとも 1 つでも満たされていない場合, エラーが出ます.
5. 3 つめの画面では以下の事項を, 誰かがゴールするまで繰り返します.
 - (a) さいころを投げる. ただし, 1 回休みの人は (d) に進む.
 - (b) 駒を進める.
 - (c) マスを確認する.
 - プラスマス** または **マイナスマス** であれば (a) に戻り, (b), (d) の順に進む.
 - 空マス** または **休みマス** であれば (d) に進む.
 - (d) ターンを終える.
6. ゴールする条件は現在地の座標がゴールの座標より大きいことです. そのため, ゴールまでのマス数より大きい目が出てもゴールとなります.

4.3 実行

4.3.1 1 画面目

1 画面目の初期状態を図 4.1 として示します。この画面では、最大 4 人のプレイヤーの

The screenshot shows a window titled "Dice" with a light gray background. It contains several input fields and a "決定" (Decide) button. Under the heading "お名前" (Name), there are four input fields labeled "1人目" (Player 1), "2人目" (Player 2), "3人目" (Player 3), and "4人目" (Player 4). Below this, under "マスの出現数" (Dice count), there are four input fields with spinners, each labeled with a type of die: "プラスマス" (Plus die), "マイナスマス" (Minus die), "空マス" (Empty die), and "休みマス" (Rest die). Each field has a description and a value of 0. At the bottom, there is a section for "プレイヤーの順番のシャッフル" (Shuffle player order) with radio buttons for "あり" (Yes) and "なし" (No). The "決定" button is located to the right of these options.

図 4.1 1 画面目の初期状態です。

プレイヤーネーム、ゴールまでに配置されるマスの数、プレイヤー順序のシャッフルを行うかを入力します。

プレイヤーネームの入力欄は 1 人目、2 人目、… となっていますが、途中に空欄があっても問題なく処理を行うことができます。

マス数は、4 種類それぞれについて 0 以上 1000 以下で指定ができ、4 種類のマスで合計 1 マス以上存在すればマップを生成できます。つまり、3 種類までは 0 を指定できます。

プレイヤー順序については、1 人目、2 人目、3 人目、4 人目の順になっています。しかし、プレイヤーの順序のシャッフルを「あり」に指定することで、順序を変更することができます。

入力例を図 4.2 として示します。

This screenshot shows the same "Dice" window as Figure 4.1, but with example values entered. The player names are "Player 1", "Player 2", "Player 3", and "Player 4". The dice counts are: "プラスマス" (20), "マイナスマス" (4), "空マス" (12), and "休みマス" (4). The "シャッフル" option is set to "あり" (Yes), indicated by an asterisk next to the label. The "決定" button is still present.

図 4.2 1 画面目の入力例です。

4.3.2 2 画面目

2 画面目の例を図 4.3 として示します。2 画面目では 1 画面目で入力した内容を確認し



図 4.3 2 画面目の例です。

ます。

入力内容に問題がなければ次の画面へ遷移するリクエストをできます。問題があれば、1 画面目に戻り、内容の修正を行うことができます。

なお、次の画面へ遷移するリクエストを行っても、必ず画面遷移をするわけではありません。いくつかの条件下ではすごろくの実行に支障をきたすため、そのような条件を排除するため、すごろくを実行できない入力があった場合は画面遷移を拒否します。

そのため、

- プレーヤーが 1 人以下である
- マス数の合計が 1 未満である
- プレイヤー順序について選択されていない

の少なくとも 1 つを満たす場合、エラーが出て、画面遷移を行うことができません。

図 4.4 として実行できないことを意味する出力の例を示します。

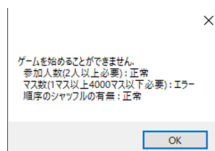


図 4.4 すごろくを実行できないことを警告する出力の例です。

なお、画面遷移に成功すると、駒を進める順番がメッセージボックスで出力されます。そ

の例を図 4.5 として示します。

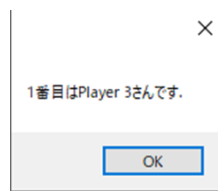


図 4.5 駒を進める順序の表示例です。

4.3.3 3 画面目

図 4.6 として 3 画面目の開始時の画面例を示します。画面上部に、順番が回ってきてい



図 4.6 3 画面目が表示された時の画面例です。

るプレイヤーのプレイヤーネームと現在地、画面右側にそのプレイヤーの先のマスの情報を表示しています。

実際にプレイするときは、このすごろくの流れ

- (1) サイコロを振る
- (2) 駒を進める
- (3) マスを確認する
- (4) ターンを終える

に対応するボタンを押します。このとき、(3) の後、駒を 1 回動かす度に止まったマスが**プラスマス**または**マイナスマス**の場合、最大で 1 回、(1) に戻ります。2 回目にサイコロを振ったときに止まったマスのイベントは無効なので、2 回目の (2) の次は (4) となります。このように、複雑になっているので実際にどのボタンを押すかは画面の上のほうに表示されます。

はじめに「**サイコロを振る**」ボタンを押します。実際に押すと、図 4.7 に示すような画面になります。

次に「**駒を進める**」ボタンを押します。実際に押すと、図 4.8 に示すように出力されます。

次に「**マスの結果を確認する**」ボタンを押します。実際に押すと、図 4.9 に示すように出力されます。

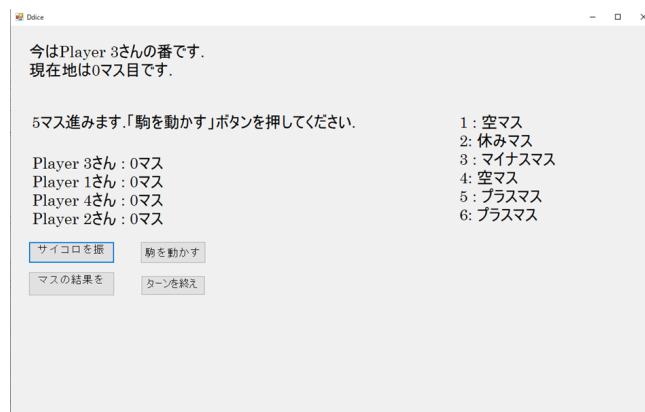


図 4.7 サイコロを振った後の出力です。

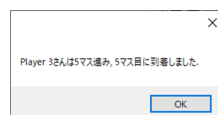


図 4.8 駒を進めた際の出力です。

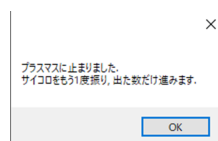


図 4.9 マスの結果を確認した際の出力です。

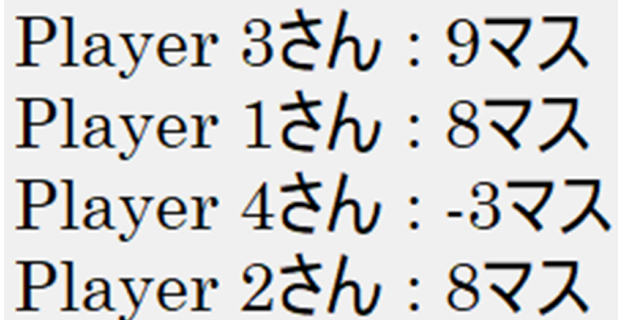
この後、止まったマスが**プラスマス**または**マイナスマス**の場合、**サイコロを振る**、**駒を進める**、**ターンを終える**の順にボタンを押します。

止まったマスが**空マス**または**休みマス**の場合、**ターンを終える**ボタンを押します。

4.4 修正するべき点

4.4.1 現在地の座標が負になりえること

現在地の座標が負になった例を図 4.10 として示します.



Player 3さん : 9マス
Player 1さん : 8マス
Player 4さん : -3マス
Player 2さん : 8マス

図 4.10 座標が負になった例です.

該当箇所のソースコードをソースコード 4.1 として示します.

```
1      tab[num].state += tab[num].diff;  
2      if (tab[num].diff > 0)  
3      { 進むとき//  
4          MessageBox.Show(tab[num].name + さんは  
              "" + tab[num].diff + マス進み", " + tab[num].state + マ  
              ス目に到着しました".");  
5      }  
6      else  
7      { 戻るとき//  
8          int tem = -tab[num].diff;  
9          MessageBox.Show(tab[num].name + さんは"" + tem + マス戻り  
              ", " + tab[num].state + マス目に到着しました".");  
10     }
```

ソースコード 4.1 Form3-progress.cs

1 行目で, `tab[num].diff` で表される, 符号付きの移動するマス数をそのまま `tab[num].state` で表される現在地の座標に加えたことが問題でした. そこで, このソースコードはソースコード 4.2 に示すようにするべきでした.

```
1      if(-tab[num].diff > tab[num].state)  
2      {  
3          tab[num].state = 0;  
4      }  
5      else  
6      {
```



```

7         tab[num].state += tab[num].diff;
8     }
9     if (tab[num].diff > 0)
10    { 進むとき//
11        MessageBox.Show(tab[num].name + さんは
            "" + tab[num].diff + マス進み
            ", " + tab[num].state + マス目に到着しました".");
12    }
13    else
14    { 戻るとき//
15        int tem = -tab[num].diff;
16        MessageBox.Show(tab[num].name + さんは"" + tem + マス戻り
            ", " + tab[num].state + マス目に到着しました".");
17    }

```

ソースコード 4.2 Form3-progress2.cs

このようにしたことで、進む場合、または現在地が戻るマス数と比較し、十分にスタートから遠い場合にはそのまま移動数を現在地に加えることで更新し、単純に戻ると、スタートを通り過ぎる、つまり現在地が負となる場合は現在地を 0 で更新すればこの問題を回避することができました。

4.4.2 さいころで 7 の目が出ること

さいころで 7 の目が出た例を図 4.11 として示します。

今はPlayer 2さんの番です。
 現在地は2マス目です。
 7マス進みます。「駒を動かす」ボタンを押してください。

図 4.11 さいころで 7 の目が出た例です。

ここでは 1 以上 6 以下の目が出ることを想定していたので、このままでは問題があります。

該当する部分のソースコードをソースコード 4.3 として示します。

```

1         int seed = Environment.TickCount;
2         Random rand = new Random();
3         int a = Convert.ToInt32(rand.NextDouble() * (5 + 1) + 1);

```

ソースコード 4.3 Form3-dice.cs

3 行目

```
Convert.ToInt32(rand.NextDouble() * (5 + 1) + 1)
```

の部分に注目すると、`NextDouble()` は閉区間 $[0, 1]$ 上の一様分布に従う乱数を返します。そのため、`Convert.ToInt32()` 内の数値は $[1, 7]$ 上の一様分布に従います。

私は `Convert.ToInt32()` が中の数値を切り捨てるものだと考えていました。しかし、実際には四捨五入するものだったので、出目 X について確率関数は理屈の上では

$$P(X = k) = \begin{cases} \frac{1}{6} & (k = 2, 3, 4, 5, 6) \\ \frac{1}{12} & (k = 1, 7) \\ 0 & (\text{otherwise}) \end{cases}$$

となっていました。

そこで、7 を 1 に書き換えれば、

- 7 の目が出ること
- すべての目が出るのが同様に確からしいわけではないこと

を解決できました。

そのソースコードは以下にソースコード 4.4 として示すようなものであるべきでした。

```
1      int seed = Environment.TickCount;
2      Random random = new Random(seed + i);
3      int a = Convert.ToInt32(random.NextDouble() * (5 + 1) +
4          1);
5      if(a == 7)
6      {
7          a = 1;
8      }
```

ソースコード 4.4 Form3-dice2.cs

4.5 終わりに

私はこれまで、授業を通じて学んだことと今回のような開発で使うことができる事柄を分けて考えていました。しかし、実際に開発を行うと、想像以上に授業で学んだことを活用できることが分かりました。さらに、授業で学んだときはどのように活用するかよく分からないままになっていましたが、自分で手を動かして開発を行うことで、学んだ内容が生かされる手法を体感することができ、より深く理解できたように感じました。

しかし、まだ素人がやっつけ仕事で開発したということが感じられるものでした。そこで、これからさらに学習を進めることで少しでも優れたものを開発できるように努めたいと思いました。

参考文献

- [1] Convert.ToInt32 メソッド Convert.ToInt32 メソッド (System)|MicrosoftLearn
- [2] NextDouble メソッド Random.NextDoubleMethod(System)|MicrosoftLearn
- [3] VSCode で C# を実行する VScode で C#を実行する#初心者\OT1\textendashQiita
- [4] ページ遷移 C#画面遷移 新しいフォームの追加|ひろにもブログ (hironimo.com)
- [5] フォーム間でのデータの受け渡し【C#】Form と Form の間で値の共有・受け渡しをする！\OT1\textendash エンジニアが送る穴倉生活のすゝめ (hiropon-progra.com)
- [6] 数値入力コントロール ソフトウェアを楽しく作ろう！プログラミング言語【C#】を学ぶ|C#数値入力コントロールの使い方 (NumericUpDown) (cozver.com)
- [7] ラジオボタン C#ラジオボタンの作成のサンプル (フォームアプリケーション)|ITSakura

編集後記

今回編集させていただいた樋口です。ここまで目を通していただきありがとうございました。執筆のスケジュールが短くなってしまいましたが、執筆者の皆様は記事を書いていただきありがとうございました。編集作業は初めてでしたが、先輩方がまとめてくださった編集作業用のデータや石倉さんのご協力のおかげで無事に完成することができました。感謝申し上げます。

令和 6 年 12 月 30 日
編集担当 樋口 舞子

Lime Vol. 64

令和 7 年 1 月 5 日発行 第 1 刷

発行 京都工芸繊維大学コンピュータ部

Web サイト: <http://www.kitcc.org/>

電子メール: question@kitcc.org
