令和 5 年 2 月 13 日
京都工芸繊維大学コンピュータ部

Lime 62

はじめに

コンピュータ部部長の中園です。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 Lime62 号を手にしてくださったことを、部を代表いたしま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木村君、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 5 年 2 月 13 日

京都工芸繊維大学コンピュータ部部長　中園 康聖

目次

はじめに iii
1　迷路ゲーム　—　石倉 朋佳 1
2　ルータ監視ハードウェア開発　—　木村 俊星 10
3　Scheme マスターを目指せ　—　中園 康聖 19
4　Unity を使ってタッチナンバーゲームを作る　—　源 晴喜 22
編集後記 30

1 迷路ゲーム

情報工学課程 1 回生 石倉 朋佳

1.1 はじめに

C 言語で CUI の迷路ゲームを作りました。このゲームではランダムに作られたステージで遊んだり、作った迷路で遊んだりすることができます。

夏休みまでの C 言語勉強会を終えた段階で、ファイルの読み書きやランダムの知識を少しかじってゲームを作ってみました。その後、プログラミングの授業で新たに学んだポインタまでの内容を踏まえて作り直すことにしました。しかし、作り直したものは完成できておらず、この記事には間に合いませんでした。授業で先生方のお話を聞いていると、作り直したいところが挙げきれないほど思い当たるのですが、成長途中の記録ということにします。まだ作り直している途中ですが、読んでいただけるとありがたいです。

1.2 概要

まず、作ったゲームの概要を説明します。基本的に、ただの単純な迷路です。地面 (マス) の状態は道、壁、スタート、ゴールがあります。壁のマスには進めず、道のマスを進んでスタートからゴールを目指します。マップは上下左右が全て壁で囲われており、マップの外には出ることはできません。道のマス上にはお宝が落ちていることがあり、拾うことができます。

top

- START (ランダムに作られる迷路でエンドレスに遊べる。迷路をクリアすればするほど、迷路が大きくなるなどの変化がある)
- FREE (自由に迷路の設定を決めて遊べる。迷路はランダムにつくられる)
- EDIT (以下の new、load、play ができる)

edit

- new (迷路を新規作成できる。作った迷路は保存できる)
- load (保存した迷路を編集することができる。編集した迷路は保存できる)
- play (保存した迷路で遊ぶことができる)

遊ぶ画面では、ターミナルに文字で描かれた 2 種類のマップが表示されます。視界のマップは、自分がいるマスとその周囲 1 マスずつの地面の状態が表示されます。記憶のマップは、これまで視界のマップに表示されたことのあるマスの地面の状態が表示されます。また、マップを囲う上下左右の壁は最初から表示されています。お宝は視界のマップに表示されます。

作る画面では、遊ぶときに迷路の中を移動するときと同様にマスを移動して、選択しているマスの地面の状態やアイテムの有無を切り替えることで、視覚的に迷路をつくることができます。作った迷路は保存し、遊ぶことができます。

移動や選択などの入力は、全てターミナルで入力を促されるので、そのままキーボードで文字を入力します。

1.3 ソースコード

この迷路ゲームよりもう少し要素の多いものを作ろうと考えていたり、最初に作ったときに理解していないことがあったりして、無駄やつっこみどころがたくさんあると思います。ソースコード全体はとても長くなってしまったので、この記事では、迷路で遊ぶところと迷路を作るところの 2 つに絞ります。ソースコード 1.1 は main 関数の外に書いたことでこの後の説明に関係があるところの一部です。ファイルの読み書きとランダムのために使った 2, 3 行目はこの記事で説明するところでは使いません。

```
1  #include <stdio.h>
2  #include <stdlib.h> // srand()、rand() 関数を使うために必要
3  #include <time.h>   // time() 関数を使うために必要
4
5  #define MAPLIMIT 32  // マップの縦横の大きさの最大値
6  enum {ROAD = 0, WALL, START, GOAL};
7
8  int layer[2][MAPLIMIT][MAPLIMIT]; // layer[0] が土地の状態、layer[1] がお宝の有無
9  char groundtype[] = { '.', 'x', 'S', 'G'};
```

ソースコード 1.1: main 関数の外 (一部)

ソースコード 1.2 は迷路で遊ぶところのソースコードです。

```
1  void play(int mapsizeY, int mapsizeX) {
2    int memomap[MAPLIMIT][MAPLIMIT];  // 記憶のマップ (まだ見ていないマスは-1)
3    int playerY, playerX;
4    for(int i = 0; i < mapsizeY; i++) {  // 記憶のマップの初期化
5      for(int j = 0; j < mapsizeX; j++) {
6        if(i == 0 || i == mapsizeY - 1 || j == 0 || j == mapsizeX - 1) memomap[i][j] =
             WALL; // マップの枠は最初から
             描画
7        else memomap[i][j] = -1;
8      }
9    }
10   for(int i = 0; i < mapsizeY; i++) {    // スタート位置のチェック
```

```
11      for(int j = 0; j < mapsizeX; j++) {
12        if(layer [0] [i] [j] == START) {
13          playerY = i;
14          playerX = j;
15          i = mapsizeY;
16          j = mapsizeX;
17          // スタートを見つけたらループを抜ける
18        }
19      }
20    }
21    int playerchar = 'o'; // プレーヤーの見た目
22    int checkquit = 0, checkclear = 0;
23    int turn = 0, step = 0, treasure = 0;
24    int moveY = 0, moveX = 0;
25
26    while(1) {
27      // プレーヤー中心の 3 × 3 のマップを更新
28      for(int i = -1; i <= 1; i++) {
29        for(int j = -1; j <= 1; j++) {
30          memomap[playerY + i][playerX + j] = layer[0][playerY + i][playerX + j];
31        }
32      }
33      // 画面描画
34      drawscreen(memomap, playerY, playerX, mapsizeY, mapsizeX, playerchar); // 記憶のマッ
            プ、視界のマップを描画する関数（別で定義している）
35      printf("\nturn:%d  step:%d    * (treasure):%d\n", turn, step, treasure);
36      printf("Input to move. Input q to quit.\n   k\n h   l\n   j\n");    // key 操作メモ
37
38      turn += 1;
39      int key;
40      do {
41        printf("input:");
42        key = getchar();
43      } while (key != 'h' && key != 'j' && key != 'k' && key != 'l' && key != 'q');
44      getchar();  // 直後のエンターを読み込む用
45      switch(key) {
46        case 'h': // 左に動きたい
47          moveX = -1;
48          playerchar = '<';
49          break;
50        case 'j': // 下に動きたい
51          moveY += 1;
52          playerchar = 'v';
53          break;
54        case 'k': // 上に動きたい
55          moveY -= 1;
```

```
56          playerchar = 'A';
57          break;
58        case 'l': // 右に動きたい
59          moveX += 1;
60          playerchar = '>';
61          break;
62        case 'q': // ゲームを終了したい
63          checkquit = 1; //終了チェック変数を 1 に変える
64          break;
65      }
66
67      if(checkquit == 1) { // 終了チェック変数が 1 のとき、ループを抜けだす
68        puts("You gave up!");
69        break;
70      }
71
72      // moveX または moveY の値に変化があるとき（移動のキーが押されたとき）
73      step += 1;
74      switch(layer[0][playerY + moveY][playerX + moveX]) { // 移動先をチェックして処理
75        case 3: // ゲームクリア
76          checkclear = 1; //クリアチェック変数を 1 に変える
77        case 2:
78        case 0: // 移動できる
79          playerX += moveX;
80          playerY += moveY;
81          break;
82        case 1:  // 移動できない
83          break;
84      }
85      treasureCheck(playerY, playerX, &treasure);  // プレーヤーの座標にお宝が落ちているか調べ、
            処理する関数（別で定義している）
86      moveY = 0;
87      moveX = 0;
88      // ともに 0 に戻しておく
89
90      if(checkclear == 1) { // クリアチェック変数が 1 のとき、ループを抜けだす
91        printf("Congratulations!\nthe whole map is below\n");
92        for(int i = 0; i < mapsizeY; i++) { // マップ全体を表示する
93          for(int j = 0; j < mapsizeX; j++) {
94            printf(" %C", groundtype[layer[0][i][j]]);
95          }
96          puts("");
97        }
98        break;
99      }
100   }
```

```
101   waitreply();
102 }
```

ソースコード 1.2: 迷路で遊ぶところ

迷路で遊ぶところではまず、記憶のマップやプレーヤーの座標の初期設定をします。その後は移動の入力のときに終了を選択するかクリアするまで、繰り返しになります。

繰り返しの中では、まずマップなどの画面を表示します。次に入力を受けつけ、入力に応じて移動の準備や終了の準備をします。その後、終了するか、移動先に進めるか、プレーヤーの座標にお宝があるか、クリアしたか、を順に調べ、それぞれ必要な処理があれば実行します。ここまで実行すると繰り返しの最初に戻ります。この繰り返しを抜けた場合、プレーヤーにメッセージなどを見てもらってから、迷路で遊ぶところは終了します。

ソースコード 1.3 は迷路を作るところのソースコードです。

```
1  void edit(int mapsizeY, int mapsizeX) {
2    int quitcheck = 0, finishcheck = 0, explanation = 1;
3    int editinglayer = 0, editingpointY = 1, editingpointX = 1; // 編集中の座標を表す
4    layerclear(mapsizeY, mapsizeX); // 多次元配列 layer の初期設定をする関数（別で定義している）
5    while(quitcheck == 0) {
6      clear();
7      for(int i = 0; i < 2; i++) {
8        printf("layer%d\n", i);
9        for(int j = -1; j < mapsizeY; j++) {
10         for(int k = -1; k < mapsizeX; k++) {
11           if(j == -1) {  // 一番上に X 軸座標を描画
12             if(k != -1) {
13                 if(k < 10) printf("%d ", k);
14                 else printf("%d", k);
15             }
16             else printf("  ");
17             continue;
18           } else if(k == -1) { // 一番左に Y 軸座標を描画
19             if(j < 10) printf("%d ", j);
20             else printf("%d", j);
21             continue;
22           } else if(i == editinglayer && j == editingpointY && k == editingpointX) {
23             // editingpoint のとき@ を描画
24             printf("@ ");
25             continue;
26           } else if(i == 0) {  // layer0 のとき
27             printf("%c ", groundtype[layer[0][j][k]]);
28           } else if(i == 1) { // layer1 のとき
29             if(layer[1][j][k] == 0) printf(". ");
30             else if(layer[1][j][k] == 1) printf("* ");
31           }
32         }
```

```
33          puts("");
34        }
35        puts("");
36      }
37      printf("editing layer:%d  @ is editing point(%d, %d)\n\n", editinglayer,
            editingpointY, editingpointX);
38      printf("input explanation\n  e   explanation(on/off)\n");
39      if(explanation == 1) {  // 詳細な操作方法の説明
40        printf("  x   wall\n  .   plain floor\n  S   start\n  G   goal\n  *   tresure\n
              \\n"
41                "  change editing layer\n  h   move editingpoint: left\n  j   move
                   editingpoint: down\n"
42                "  k   move editingpoint: up\n  l   move editingpoint: right\n"
43                "      quit without save (push space once and push Enter once)\n  _
                   finish!\n");
44      }
45      printf("\ninput:"); // 入力を促す
46
47      switch(getchar()) {
48        case 'x':
49          getchar();
50          if(editinglayer == 0) layer[0][editingpointY][editingpointX] = WALL;
51          break;
52        case '.':
53          getchar();
54          if(editinglayer == 0) layer[0][editingpointY][editingpointX] = ROAD;
55          else if(editinglayer == 1) layer[1][editingpointY][editingpointX] = 0;
56          break;
57        case 'S':
58          getchar();
59          if(editinglayer == 0) layer[0][editingpointY][editingpointX] = START;
60          break;
61        case 'G':
62          getchar();
63          if(editinglayer == 0) layer[0][editingpointY][editingpointX] = GOAL;
64          break;
65        case '*':
66          getchar();
67          if(editinglayer == 1)layer[1][editingpointY][editingpointX] = 1;
68          break;
69        case '\n':    // editinglayer を一つ次の番号のものにする（末尾は 0 に戻る）
70          if(editinglayer < 1) editinglayer += 1;
71          else editinglayer = 0;
72          break;
73        case 'h': // 左に動く
74          getchar();
```

```
75          if(editingpointX > 1) editingpointX -= 1;
76          break;
77        case 'j': // 下に動く
78          getchar();
79          if(editingpointY < mapsizeY - 2) editingpointY += 1;
80          break;
81        case 'k': // 上に動く
82          getchar();
83          if(editingpointY > 1) editingpointY -= 1;
84          break;
85        case 'l': // 右に動く
86          getchar();
87          if(editingpointX < mapsizeX - 2) editingpointX += 1;
88          break;
89        case ' ': // 保存せずに終わる。
90          getchar();
91          quitcheck = 1;
92          break;
93        case '_': // 保存して終わる。
94          getchar();
95          finishcheck = 1;
96          quitcheck = 1;
97          break;
98        case 'e': // 詳細な操作方法の説明の ON/OFF 切り替え
99          getchar();
100         explanation *= -1;
101         break;
102       default:
103         getchar();
104         break;
105     }
106   }
107 }
```

ソースコード 1.3: 迷路を作るところ

迷路を作るところではまず、編集しているところの座標を表す変数や編集で使う多次元配列 layer の初期設定をします。その後は保存せずに終了か完了を選択するまで繰り返しになります。繰り返しの中では、まずマップなどの画面を表示します。次に入力を受けつけ、適切な入力ならば処理を実行します。この繰り返しを抜けた場合、そのまま迷路を作るところは終了します。

1.4 プレイ画面

図 1.1 は、迷路で遊んでいる途中の画面です。プレーヤーが移動した分だけ記憶のマップが更新され、視界のマップにはお宝が落ちているのが見えます。図 1.2 は、迷路を作っている途中の画面です。上の layer0 でマップの土地の状態を、下の layer1 でお宝の有無を編集します。 @ が今編集できる座標です。

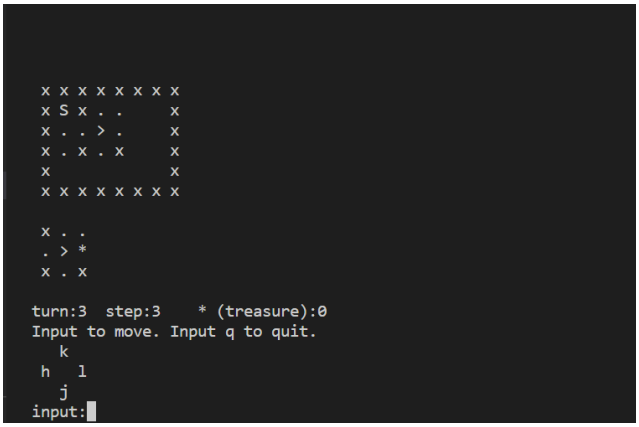


図 1.1: 迷路で遊んでいる途中の画面

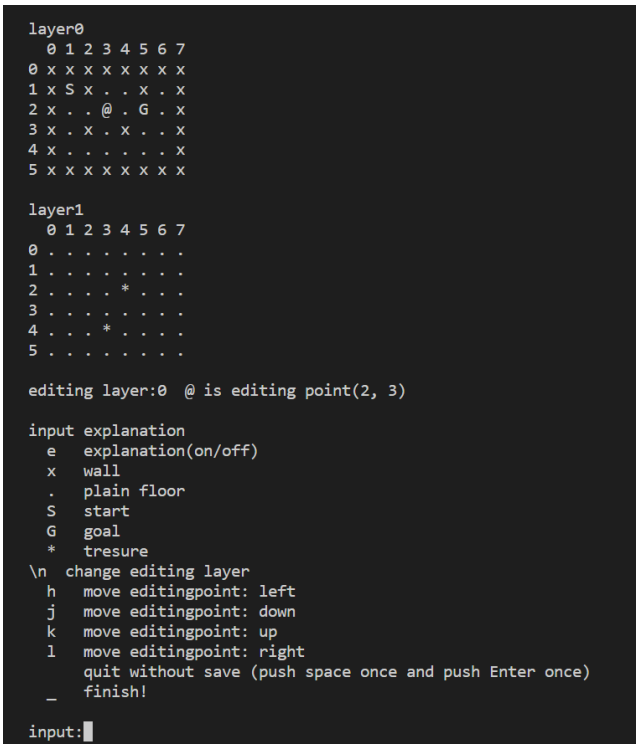


図 1.2: 迷路を作っている途中の画面

1.5　おわりに

まず、初めて C 言語でゲームを作って、夏休みまでの C 言語勉強会の知識と少しの追加の知識でも簡単なゲームが作れることに驚きました。C 言語勉強会で学んだことも、プログラミングの授業で新たに学んだことも、最初はなかなか覚えられませんでしたが、拙くても実際に使ってゲームを作ってみることで覚えることができたり、自分が理解できていないところに気付くことができたりして、良かったです。

このゲームは一旦完成とした後、約 1 ヶ月ずっと放置してしまったため、読み返すことからとても大変でした。できることを少しずつ増やして付け足しながら作っていたので、全体像を見直して最初から作り直すことになりました。そこに時間を取られてしまい、ここで説明した部分は結局あまり作り直しの手が入っていないことに心残りがあります。次はしっかり納得のいく完成まで仕上げたいと思います。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] 柴田望洋 (2022) 新・明解 C 言語 入門編 第 2 版. SB クリエイティブ株式会社.

2 ルータ監視ハードウェア開発

情報工学課程 3 回生 木村 俊星

2.1 前置き

- この記事は，開発対象をよく理解できていなかった人間が，精一杯調べながら執筆した記事であるため，デタラメな内容も多く含まれていると思う．この記事を読む過程で疑問を感じた点はぜひともご指摘頂きたい．KITCC のメールアドレスは編集後記のページに書かれている．
- この記事を書く頃には既に納品していたため，実際の実行結果や実物の写真の掲載は叶わなかった．

2.2 成果物の要約

2022 年の 10 月上旬に，KITCC の OB さんからの依頼を受けて，ADSL ルータを監視するハードウェアと，それを制御する簡単なプログラムを開発した．“監視する” をもう少し具体的に言うと，定期的にインターネットへの接続が異常でないか確認し，接続が落ちていたら電源を切って数分後に ON にすることである．今年の部誌には，このときに得た知識や成果物の内容をまとめてみようと思う．

作成したハードウェアのの部品配置図[*1]を図 2.1 に示す．PC サーバで実行するプログラムで USB シリアル変換ボードの RTS 線を制御し，リレーを経由してルータに電源を供給するか否かを切り替えるという仕組みである．回路を構成する各部品については 2.3 節で説明する．

[*1] 物理的な部品の配置を示した図

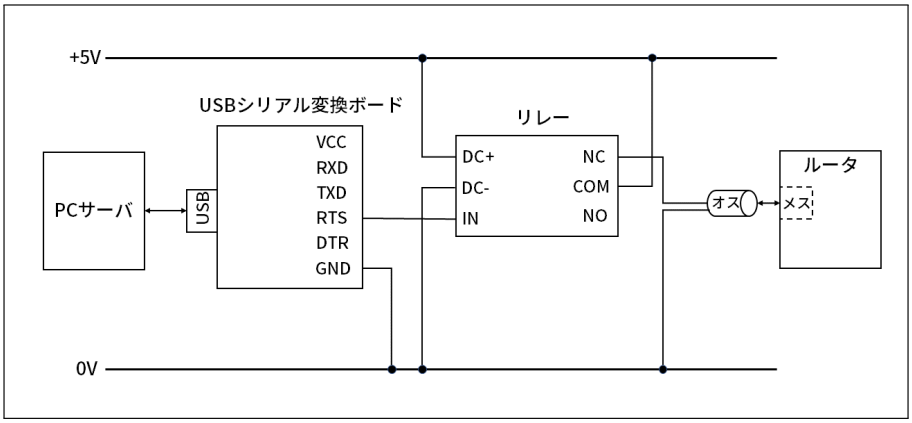


図 2.1: 回路全体の部品配置図

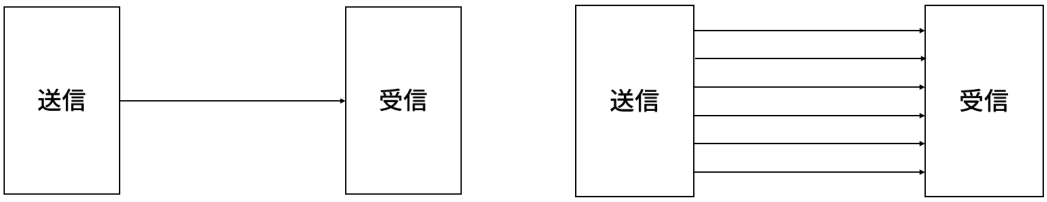
2.3 ハードウェア

2.3.1 USB シリアル変換ボード

USB シリアル変換ボードは，RS-232C という規格に準拠した機器と USB という別の規格に準拠した機器の間で通信を行う際に，電気信号を変換するための部品である．まず，シリアル通信とは何かという話から始めて，シリアル通信の規格の変遷を辿ることで変換の必要性を説く．

シリアル通信とは

パソコンと周辺機器，あるいはパソコンとパソコン間で通信を行う方式は，大きく**シリアル通信**と**パラレル通信**に分けられる．シリアル (serial) は “直列” という意味であり，1 本の信号線を使用して 1 ビットずつ連続して伝送する．パラレル (paralell) はシリアルの対義語で “並列” という意味であり，複数の信号線を使用して複数のビットを一度に伝送する．シリアル通信とパラレル通信のイメージを図 2.2 に示す．



(a) シリアル通信　　(b) パラレル通信

図 2.2: シリアル通信とパラレル通信のイメージ

当初は複数の信号を一度に伝送できるパラレル通信の方が高速とされていたが，高速化を突き詰めていくにつれて信号間のタイミングをとるのが難しくなり，現在はシリアル通信で高速化を図るのが主流である．

シリアル通信の規格

有線の通信規格にはケーブルや端子の差し込み口の形状や，ケーブルの中を通す信号のパターンなどが細かく規定されている．この規格を守ることで，異なるメーカの機器を難なく交換できたり，プリンタとスキャナのように異なる用途の機器で同じケーブルを共用できたりといった互換性が保たれている．

シリアル通信の現在最も標準的な規格は USB であるが，USB が登場する以前はモデム[*2]には RS-232C，ハードディスクやスキャナには SCSI [*3]，キーボードやマウスには PS/2 といったように，機器の種類ごとに異なる規格が定められており，これらのコネクタとケーブルは現在の USB よりもずっと大きなものであった．パソコンと周辺機器を接続するには，当然パソコン側にも同じポートを搭載しなければならず，小型化を妨げる要因になっていた．これらの規格を統一することを主な目的として考え出された規格が USB であり，Universal という規格名の所以でもある．

ネットワークの普及でモデムがほとんど使われなくなったことや，より汎用性の高い USB が普及したことで，最近のパソコンの多くはシリアルポート (RS-232C ポート) を備えていない．シリアルポートを備えていないパソコンと RS-232C に対応した機器で通信を行うには，各規格で定められた信号パターンを変換する必要がある．変換を実現する方法には，ソフトウェアによってパソコンに仮想的なシリアルポートを形成する方法と，今回のように変換ボードや変換アダプタなどのハードウェアを用いて変換する方法がある．

2.3.2 リレー

リレーとは外部からの電気信号によって回路の開閉を切り替える部品であり，電磁石を利用した**有接点リレー**と半導体素子を利用した**無接点リレー**に分類される．今回の開発で使用した有接点リレーは**メカニカルリレー**とも呼ばれ，物理的にスイッチを切り替えるためカチッと音がするのが特徴である．

メカニカルリレーは，電気信号を機械的な動きに変えるコイル部と，回路を開閉する接点部から構成される．接点部の構造にも分類があり，電流を流すと回路が閉じる **a 接点**，電流を流すと回路が開く **b 接点**，電流を流さないと b 接点に，電流を流すと a 接点に接続する **c 接点**がある．

メカニカルリレーの動作原理は [4]，a ～ c 節点の構造と動作については [5] が個人的に分かりやすかったため，気になる方にはこれらの記事を読んで頂くことにして，本稿ではこれ以上詳説しない．

2.3.3 回路全体の動作

ルータの接続異常を検知してから電源の供給を断ち，ルータを再起動するまでのまでの一連の流れを以下に箇条書きで記す．

1. 通常は RTS 線を OFF にしておく．

[*2] 変調器を意味する modulator と復調器を意味する demodulator をつなぎ合わせた造語．
デジタル信号とアナログ信号を相互に変換する機器を指す．

[*3] Small Computer System Interface の略でスカジーと読む．

リレーの COM と NC が接続され，リレーを介してルータへと電源が供給される．

2. 数分おきにルータの接続を確認する．
   （a）接続が正常であれば 2. を繰り返す．
   （b）接続が落ちていたら RTS 線を ON にする．
   リレーの COM と NO が接続され，ルータへの電源供給が絶たれる．
3. 数分間待機した後，RTS 線を OFF に戻して 2. へ戻る．．

上記の動作をプログラムが強制終了されるまで永遠に続ける．

2.4 ソフトウェア

PC サーバで実行するプログラムには 2 つの仕事がある．

- ルータの接続状態を定期的に確認する．
- RTS 線を制御する．

それぞれの具体的な処理をソースコードを示しながら説明する．

2.4.1 開発環境

```
1  $ cat /etc/lsb-release
2  DISTRIB_ID=Ubuntu
3  DISTRIB_RELEASE=22.04
4  DISTRIB_CODENAME=jammy
5  DISTRIB_DESCRIPTION="Ubuntu 22.04.1 LTS"
6
7  $ cat /proc/version
8  Linux version 5.15.0-56-generic
9      (buildd@lcy02-amd64-004)
10     (gcc (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0,
11      GNU ld (GNU Binutils for Ubuntu) 2.38)
12     #62-Ubuntu SMP Tue Nov 22 19:54:14 UTC 2022
13
14 $ gcc --version
15 gcc (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0
16 Copyright (C) 2021 Free Software Foundation, Inc.
17 This is free software; see the source for copying conditions.
18 There is NO warranty; not even for MERCHANTABILITY or FITNESS
19  FOR A PARTICULAR PURPOSE.
```

2.4.2 ルータの監視

ルータの接続状態を確認する関数を図 2.3 に示す.

```
1  int check_connection(){
2      // ping コマンドの実行
3      // パケットを 4 回送信，コマンドの起動時間は 10 秒
4      int result = system("ping -c 4 -w 10 www.example.com");
5
6      // 子プロセスが正常に終了している場合のみステータスを取得
7      if (WIFEXITED(result)) {
8          return(WEXITSTATUS(result));
9      } else {
10         return(-1);
11     }
12 }
```

図 2.3: ルータを接続状態を確認する関数

ルータの生死を確認するには ping コマンドを用いた．ping は指定された宛先にパケットを送り，正しく届いて返答が行われるかを確認するコマンドである．オプションでコマンドの動作を詳細に指定でき，今回のプログラムでは-c オプションと-w オプションでパケットの送信回数と応答の待機時間を指定した．

C 言語のプログラムでは system() 関数を利用して UNIX のコマンドを実行できる．この関数は stdlib.h にて

```
int system(const char *command);
```

と宣言されており，文字列として受け取ったコマンドを子プロセスを生成して/bin/sh -c command の形で実行する．system() 関数の man ページ [7] によれば,

- command が NULL のときはシェルが利用可能ならゼロ以外の値を返し，利用不可なら 0 を返す.
- コマンドを実行する過程でエラーが発生した場合は -1 を返す.
- コマンドを実行できた場合はそのコマンドの終了ステータスを返す.

最後のコマンドの終了ステータスは以下のマクロを使って検査できる.

WIFEXITED
: 子プロセスが正常に終了した場合に真を返す．“正常に” とは exit か_exit() が呼び出された場合，もしくは main() から復帰した場合を指す.

WEXITSTATUS
: 子プロセスの終了ステータスを返す．終了ステータスは exit() や_exit() の呼び出し時に渡された値，もしくは main() の return 文の引数として指定された値である．このマクロを使用する

のは WIFEXITED が真を返した場合だけにすべきである．

実験的に ping を system() 関数で実行し，正しく応答が返ってきた時には終了ステータスが 0 になることが分かったため，図 2.3 の関数では，WIFEXITED マクロが偽であった場合の返り値を −1 としている．[*4]

2.4.3 RTS 線の制御

図 2.3 の check_connection() 関数からの返り値に応じて RTS 線を制御するメイン関数を図 2.4 に示す．紙幅を有効活用するために複数の#include 指令を 1 行にまとめて記述している．

[*4] この文脈では 0 以外の値であれば何でもよい．

```c
#include <fcntl.h>      #include <stdio.h>      #include <stdlib.h>
#include <sys/ioctl.h>  #include <termios.h>    #include <unistd.h>

#define PERIOD 120
#define RESTRAT_SEC 60

int main(){
    int fd = open("/dev/ttyUSB0", O_RDONLY); // 読み出し専用でオープン
    if (fd < 0) {
        printf("open error\n");
        exit(EXIT_FAILURE);
    }

    int serial = TIOCM_RTS;  // ioctl で設定するビット

    printf("BIS: RTS=Lo\n");
    ioctl(fd, TIOCMBIS, &serial);

    while(1){
        sleep(PERIOD);

        int status = check_connection(); // ルータの接続確認
        if(status != 0){
            ioctl(fd, TIOCMBIC, &serial); // RTS 線を ON にする
            printf("BIC: RTS=Hi\n");

            sleep(RESTRAT_SEC);

            ioctl(fd, TIOCMBIS, &serial); // RTS 線を OFF にする
            printf("BIS: RTS=Lo\n");
        }
    }
    return(0); // never reached
}
```

図 2.4: メイン関数

Linux のパソコンに USB シリアル変換機器を接続すると，

```
/dev/ttyUSB*
```

のように認識される．*は挿した順番によって決定される連番である．これは**スペシャルファイル**と呼ばれるファイルであり，カーネル内部の機能を呼び出すインタフェースとして機能する．プログラムは，スペシャルファイルやシステムコールによってカーネルを呼び出し，カーネル内のデバイスドライバを経由してハードウェアにアクセスする．

RTS 線を制御するにはスペシャルファイルを読み書きできる必要があるが，一般ユーザにはデフォルトでこの権限は付与されていない．一般ユーザとして実行する場合は，/etc/group ファイル[*5]の dialout エントリ[*6]に当該ユーザを追記する．ただし，このファイルの編集には root 権限が必要である．/etc/group の変更は当該ユーザが新たにログインした時に反映されるため，既にログインしているユーザには変更が反映されないことに注意する．

スペシャルファイルを読み書きするには ioctl というシステムコールを発行して，スペシャルファイルを管理しているデバイスドライバに命令を送る．図 2.4 のメイン関数では，USB シリアル変換ボード以外はパソコンに接続されていないという前提で/dev/ttyUSB0 を open() し，ioctl() 関数によってスペシャルファイルに値を書き込んでいる．

ところで，動作確認を終えてから気が付いたのだが，読み出し専用でオープンしているのに，なぜ値を書き込めたのだろうか?

2.5 開発を終えて

私は電子工作の経験に乏しく，ハードウェアについての知識はほとんど無い．はんだづけの経験はあったが，いずれも回路の設計は済んだ状態から部品を基板の上にはんだ付けするという半ば作業と化した工程であった．

今回の開発は，各部品をどう接続するかを自ら考える初めての機会となり，開発の手順を組み立てるまでには苦労した．開発を終えることができたのは，同期の井上蒼士 氏の助言があったからである．物分かりの悪い私の相談に根気強く付き合ってくれた井上氏と，開発の機会を与えて下さった OB さんにこの場を借りて深謝する．

参考文献

[1] シリアル通信の仕組み徹底解説．http://serialcomm.info/

[2] Think IT．「シリアル通信って何?」．https://thinkit.co.jp/story/2015/04/15/5791

[3] Rentec Insight．「意外と知らない USB の基礎知識」．
https://go.orixrentec.jp/rentecinsight/measure/article-6

[4] オムロン電子部品サイト．「リレーの基礎知識: 基礎編」．
https://components.omron.com/jp-ja/products/basic-knowledge/relays/basics

[*5] Linux では/etc/group でユーザがどのグループに入っているかを管理する．

[*6] USB を含むシリアルポートへのアクセス権限を持つグループ

[5] Electorical Information.「a 接点・b 接点・c 接点の違いや記号について」.
https://detail-infomation.com/relay-contact/
[6] @IT.「ping ～ネットワークの疎通を確認する」.
https://atmarkit.itmedia.co.jp/ait/articles/0108/30/news002.html
[7] Ubuntu Manpage: system - シェルコマンドの実行
https://manpages.ubuntu.com/manpages/impish/ja/man3/system.3.html
[8] hassiweb's programming.「Linux でシリアル通信を行う方法」.
https://hassiweb-programming.blogspot.com/2020/02/serial-comm-on-linux.html
[9] Ubuntu Manpage: tty_ioctl - 端末とシリアルラインの入出力制御
https://manpages.ubuntu.com/manpages/focal/ja/man4/tty_ioctl.4.html

3 Schemeマスターを目指せ

情報工学課程 3 回生 中園 康聖

3.1 はじめに

今まで手続き型言語ばかり学んできたので、この機会に関数型言語を学ぼうと思いました。

3.2 Scheme を選んだ理由

Scheme は実用的な言語ではないのですが、有名なプログラミングの教科書 SICP でも使われているため、マスターしておくとかなり役に立ちます。完全な関数型言語ではないですが、まずここから勉強していきます。

3.3 学んだ内容

3.3.1 いざ Scheme を始めよう

今回は Scheme を Ubuntu で学ぶので Gauche というパッケージを使います。まず以下のようなコマンドを実行します。

```
1 sudo apt install gauche
```

ソースコード 3.1: Gauche のインストール

パスワードを打ち込み管理者権限でこのコマンドを実行すると Gauche がインストールできます。

3.3.2 電卓

まずは簡単な計算から実行していきます。

```
1 gosh
```

ソースコード 3.2: Gauche の起動

このコマンドを実行すると Gauche が使用できるようになります。まずは簡単な計算を行います。

```
1 (+ 1 2)
```

ソースコード 3.3: 加算の命令

これは足し算で結果は 3 が返ってきます。+ の部分は関数名を表していて、1 と 2 は引数となります。関数型言語はこのような関数の組み合わせで構成されます。他にも以下のような命令があります。

```
1 (+ 1 2) ; 加算の命令
2 (- 2 1) ; 減算の命令
3 (/ 4 2) ; 除算の命令
4 (* 2 3) ; 乗算の命令
5 (quotient 7 3) ; 商を求める
6 (modulo 7 3) ; 余りを求める
```

ソースコード 3.4: Scheme の基本計算の命令

3.3.3 リストを作ろう

リストの構成要素はコンスセルといいます。コンスセルはアドレスを 2 つ収納したメモリ領域で 1 つめのアドレスを car 部、2 つめのアドレスを cdr 部といいます。

$$(cons12)$$

```
1 (cons 1 2)
```

ソースコード 3.5: コンスセルの生成命令

このコマンドでコンスセルが生成されます。このコンスセルを組み合わせていくことでリストができます。

```
1 (list 1 2 3 4)
```

ソースコード 3.6: リストの生成命令

list 関数は任意個の引数をとり、それらからなるリストを返します。

3.3.4 関数をマスターしよう

自分で関数を作る方法としては lambda 関数を使う方法と使わない方法の二つがあります。$define$ は 2 つの引数をとります。この関数は最初の引数の名前の変数を宣言すると同時にその値を 2 番目の引数の値にします。これを使って、手続きに関数名を付けます。手続きは $lambda$ という特殊形式を用いて定義します。$lambda$ は 1 つ以上の引数をとり、最初の引数は定義する手続きがとる引数のリストです。

```
1 (define fhello (lambda () "Hello World))
```

ソースコード 3.7: lambda を使った関数の定義

このコードは”Hello World”と表示される $fhello$ という関数です。$lambda$ を使わず関数の定義を省略すると、以下のコードのようになる。

$$(define(sumabc)(+abc))$$

```
1 (define (sum a b c) (+ a b c))
```

ソースコード 3.8: lambda を省略した関数の定義

この関数は 3 つの引数の和を返す関数です。

3.4 おわりに

Scheme を学んでいると、コンピュータサイエンスを学ぶことになります。遅延評価や、末尾再帰など手続き関数ばかり学んでいる人にとっていい刺激になるのではなるのではないでしょうか。

4 Unityを使ってタッチナンバーゲームを作る

情報工学課程 3 回生 源 晴喜

4.1 はじめに

Unity を今まで使ったことがなかったので、一度触れてみたいと思い、Unity でゲームを作ってみました。

初見で複雑なゲームを作ることは難しいと考え、操作が単純そうなタッチナンバーゲームを作ることにしました。

4.2 プログラムの概要

このゲームでは、図 4.1 左側のようにランダムに数字が配置されたボタンが用意され、より早く 1 から順番にボタンを押していきます。正しいボタンを押すとそのボタンは無効化されて暗くなって押せないようになり、ある程度の数のボタンが正しく押されると、ボタン上の数字が新しい数字に更新され、再びボタンが押せるようになります。最後の数字までボタンが押されると、図 4.1 右側のように、スタート時からかかった時間が表示されます。

図 4.1: タッチナンバーゲームの実行例

4.3 プログラムの内容

本来はスタート画面なども用意していますが、ここではゲーム本編部分のみを紹介し、それ以外の部分の説明は割愛させていただきます。プログラムの作成に当たっては、[1]、[2]、[3] を参考にしました。

4.3.1 処理部分のソースコード

ボタンを押されたときの処理を記述したソースコード Pushed.cs と、ボタン上の数字の管理やその数字の更新、時間計測などの処理を記述したソースコード SetNumber.cs を作成しました。

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using TMPro;
6  using System;
7
8  public class Pushed : MonoBehaviour
9  {
10     public void ButtonPushed(Button pushedButton)
11     {
12         int buttonNum = int.Parse(pushedButton.name.Replace("Button", "")); //管理番号
13         int pushedNum = int.Parse(pushedButton.GetComponentInChildren<TextMeshProUGUI
               >().text);
14         int correctNum = SetNumber.GetNextNumber();
15         if(pushedNum == correctNum)
16         {
17             pushedButton.interactable = false;
18             SetNumber.UpdateNextNumber(buttonNum);
19         }
20     }
21 }
```

ソースコード 4.1: Pushed.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using TMPro;
6  using System;
7  using System.Linq;
8
9  public class SetNumber : MonoBehaviour
10 {
11     static Button btn1;
12     static Button btn2;
13     static Button btn3;
14     static Button btn4;
15     static Button btn5;
16     static Button btn6;
17     static Button btn7;
18     static Button btn8;
19     static Button btn9;
20     static Button btn10;
21     static Button btn11;
22     static Button btn12;
23     static Button btn13;
24     static Button btn14;
25     static Button btn15;
26     static Button btn16;
27     static Button btn17;
28     static Button btn18;
29     static Button btn19;
30     static Button btn20;
31     static Button btn21;
32     static Button btn22;
33     static Button btn23;
34     static Button btn24;
35     static Button btn25;
36     public GameObject returnMenubtn;
37     public TextMeshProUGUI nextNumTxt;
38     public TextMeshProUGUI timeTxt;
39     private float timeScore;
40
41
42
43     public Button[] Buttons = new Button[] { btn1, btn2, btn3, btn4, btn5, btn6, btn7,
           btn8, btn9, btn10, btn11, btn12, btn13, btn14, btn15, btn16, btn17, btn18,
          btn19, btn20, btn21, btn22, btn23, btn24, btn25 };
```

```
44      private static int count;
45      private static List<int> buttonList = new List<int>();//無効化したボタンリスト
46      const int maxNum = 50;
47      private int nowMax;
48      private static int nextNum;
49      private const int addTiming = 10;
50      bool finishFlag;
51
52      // Start is called before the first frame update
53      void Start()
54      {
55          nextNum = 1;
56          nowMax = 0;
57          int startSize = 25;//ボタン数
58          count = 0;
59          buttonList.Clear();
60          finishFlag = true;
61          timeScore = 0;
62          buttonList.AddRange(Enumerable.Range(0, startSize).ToList());
63          AddNextButtons(startSize);
64      }
65
66      // Update is called once per frame
67      void Update()
68      {
69          if (nextNum > maxNum)
70          {
71              if (finishFlag)
72              {
73                  nextNumTxt.text = "Finish!";
74                  timeTxt.text = "Time : " + timeScore;
75                  returnMenubtn.SetActive(true);
76                  finishFlag = false;
77              }
78          }
79          else
80          {
81              nextNumTxt.text = "" + nextNum;
82              timeScore = timeScore + Time.deltaTime;
83              if (count >= addTiming)
84              {
85                  AddNextButtons(addTiming);
86                  count = 0;
87              }
88          }
89      }
```

```
90
91      public static void UpdateNextNumber(int buttonNum)
92      {
93          nextNum++;
94          count++;
95          buttonList.Add(buttonNum - 1);
96      }
97
98      public static int GetNextNumber()
99      {
100         return nextNum;
101     }
102
103     private void AddNextButtons(int size)
104     {
105         int diff = maxNum - nowMax;
106         int addSize = (size < diff) ? size : diff;
107         int[] addNumAry = Enumerable.Range(nowMax+1, addSize).ToArray();
108         int temp;
109         System.Random rand = new System.Random();
110         nowMax = nowMax + addSize;
111         int i;
112         while (addSize > 0)
113         {
114             i = rand.Next(0, addSize);
115             addSize--;
116             temp = addNumAry[i];
117             addNumAry[i] = addNumAry[addSize];
118             Buttons[buttonList[addSize]].GetComponentInChildren<TextMeshProUGUI>().
                    text = "" + temp;
119             Buttons[buttonList[addSize]].interactable = true;
120         }
121         buttonList.Clear();
122     }
123 }
```

ソースコード 4.2: SetNumber.cs

Pushed.cs の ButtonPushed() 関数では、押されたボタン上の数字と次に押されるべき数字を比較し、一致していたらそのボタンを押せないようにして、次に押されるべき数字を更新する UpdateNextNumber() 関数を呼び出す処理を行っています。

SetNumber.cs の Start() 関数では、ゲームスタート時に各変数を初期化し、AddNextButtons() 関数を呼び出すことで、それぞれのボタンに対して 1 から 25 までの数字をランダムに振り分けてセットする処理を行っています。

また、Update() 関数は 1 フレームごとに実行する関数で、常に正確な「次に押すべき数字」を表示し、

ゲームスタートからの経過時間の加算を行います。もしボタン上の数字を更新するのに十分な数のボタンを押されていたら、ボタン上の数字を更新して有効化を行う AddNextButtons() 関数を呼び出します。また、もし直近で正しく押された数字が最後の数字になっていた場合は、画面中に”Finish!”の文字と、それまでにかかった時間を表示します。

UpdateNextNumber() 関数では、次に押されるべき数字を 1 増やし、現在正しく押されて無効化されているボタンの数を 1 増やし、無効化されたボタンリストの buttonList に引数のボタンを追加する処理を行っています。

GetNextNumber() 関数は、次に押されるべき数字を返す関数です。

AddNextButtons() 関数では、buttonList に格納された無効化されているボタンに対して、addSize 個分の次の数字をランダムに振り分けて更新し、有効化する処理を行っています。

4.3.2　UI 部分の作成

UI 部分の作成は Unity では GUI で直感的に行えるので、図 4.2 のように、Canvas 上に Panel を設置し、Panel の上に、25 個のボタンを配置した ButtonsPanel、次に押すべきボタンの数字を示すテキストの NextNumber、”Next”の文字を表示するテキストの Next、初期は空文で、ゲーム終了時にかかった時間を表示するテキストの TimeText (図 4.2 中央の黄色の枠線部) を配置しました。

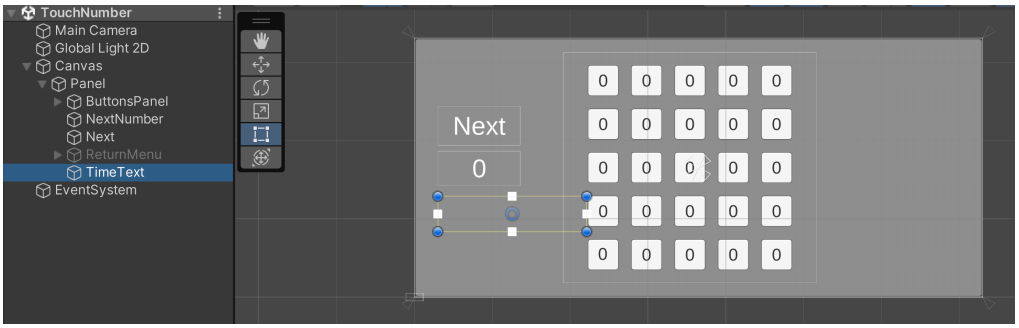


図 4.2: UI 部品の配置

4.3.3　スクリプトとオブジェクトとの紐づけ

スクリプトとオブジェクトとの紐づけとして、Panel に `SetNumber.cs` をアタッチし、`SetNumber.cs` 中のオブジェクト nextNumTxt、timeTxt、及び Button 型配列 Buttons の各要素をそれぞれ、UI 部品の NextNumber、TimeText、及び ButtonsPanel 中のそれぞれのボタンと対応付けました。

また、ButtonsPanel 中のボタンそれぞれに `Pushed.cs` をアタッチし、ボタンのクリック時に、引数にクリックされたボタン自身を適用して ButtonPushed() 関数を呼び出すように設定しました。

4.4 おわりに

今回、ひとまず動く物は作れましたが、特に `SetNumber.cs` に関しては、もっとシンプルにコードが書けたのではないかなと思いました。

また、Unity の使い方をある程度理解するまでは大変でしたが、一度理解できると GUI で直感的に作れる部分も多く、自分の作りたいものを表現しやすいように感じました。

Unity はチュートリアルが充実していて、解説しているサイトなども多いので、この記事を読んでもし気になった方は一度触ってみてもいいかもしれません。

参考文献

[1] Introduction to Roll-a-ball - Unity Learn
https://learn.unity.com/tutorial/introduction-to-roll-a-ball#

[2] 【unity 入門】interactable を使った！一度だけ押せる Button の作り方 — もぎブログ (mogi0506.com)
https://mogi0506.com/unity-button-interactable/

[3] Unity5 の「ボタンのテキスト」を取り出したい (teratail.com)
https://teratail.com/questions/73655

編集後記

編集担当の木村です。今回で 62 冊目の Lime になります。

この部誌は外部の方にコンピュータ部の活動を知って頂く機会であると共に、部員が自ら取り組んだことをアウトプットする貴重な機会でもあります。記事を書いてくれた部員は、執筆の過程で自分の知識の抜けや漏れに気づいたり、新たな疑問点が出てくることもあって、よい成長の機会になったのではないかと思います。各部員が現時点の理解に基づいて記事を書いているので、不正確な記述もあったと思いますが、間違いは下記のメールアドレスまでご指摘頂けると嬉しいです。

最後になりましたが、ここまで目を通していただき、ありがとうございました。

令和 5 年 2 月 13 日
編集担当　木村 俊星

Lime Vol. 62
令和 5 年 2 月 13 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