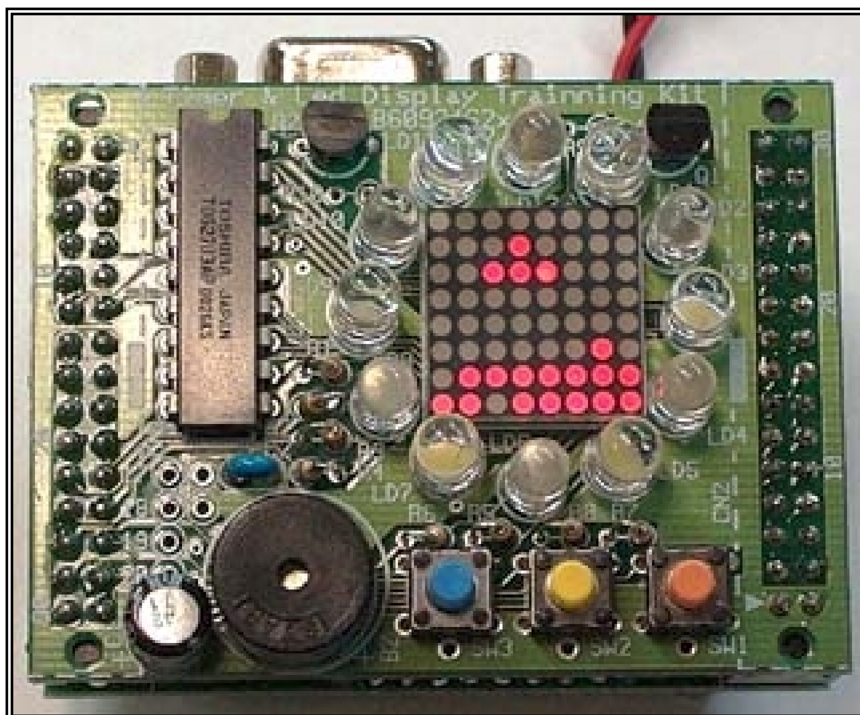
TK-3687/TK-3687mini トレーニングオプション

テトリス&LEDディスプレイ

Version　1.02



　一つのアプリケーションを設計しまとめあげる方法は参考書を読めば身につくというものではありません。プログラムの文法は知っていても，大きなプログラムを作ろうとするとどこから手をつけたらよいかわからない，ということはよくあります。このマニュアルではタイマ&LED ディスプレイでテトリスを作ることで，プログラムの設計や考え方を学習します。

目次

1　テトリスの仕様 ……………………………………………… P. 1
2　データの設計 ……………………………………………… P. 3
3　メインプログラム …………………………………………… P. 5
4　ゲームアクションのプログラム ……………………………… P. 7
5　プログラムを改造する ……………………………………… P. 15
　付録 ……………………………………………………………… P. 16

(株)東洋リンクス

1 テトリスの仕様

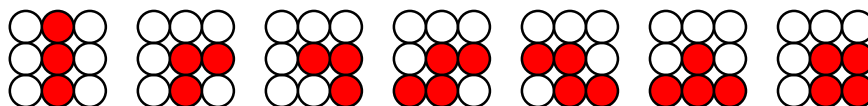
テトリスは，いわゆる「落ちゲー」と呼ばれる種類のミニゲームです。ルールは単純ですが，なぜかはまってしまうようですね。有名なゲームなのでご存知の方が多いと思います。もしどんなゲームか知らないので調べてみたいと思う方は，インターネットで「テトリス」を検索してみてください。パソコン上で動くプログラムがいくつも見つかるはずです。中にはインストールしなくてもブラウザ上で動くテトリスもあります。実際に動かしてみるとゲームのイメージがつかめるでしょう。

さて，プログラムを作るにあたり最初に行なう作業は，どんなプログラムにするか仕様を決める，ということです。さすがにこれがないと先に進むことができません。では，タイマ&LED ディスプレイ版テトリスの仕様を考えてみましょう。

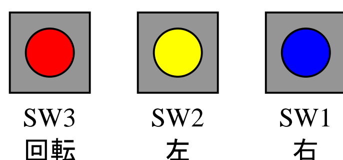
ゲーム画面の広さは LED ディスプレイで決まってしまいます。というわけで 8×8 ドットにします。上からブロックが出現し，一定時間毎に下に落ちます。下に落ちることができなくなったらそこで固定され次のブロックが再び上から現れます。

ブロックが固定されたときに横一行ブロックがそろっているとその行が削除され，消えた行の上にブロックがあるときは消えた行数分だけ下がります。ブロックを消すことができずに一番上まで到達したらゲームオーバーです。

落ちてくるブロックの形は次の 7 種類で，どのブロックにするかは乱数を使ってランダムに決定します。



ゲームユーザが操作できるのは今落ちているブロックです。ブロックを左右に移動させたり，ブロックを回転して向きを変えたりできます。タイマ&LED ディスプレイのスイッチ SW1～3 で操作します。



ゲームなので点数も付けましょう。1 行削除すると 1 点とします。ただし，1 行づつ削除するよりも 2 行や 3 行まとめて削除するほうが難易度が高いので，2 行まとめて削除したときは 4 点，3 行まとめて削除したときは 9 点にします。（点数はまとめて削除した行数×行数で計算します）

ゲームが進むにつれて難易度を上げます。0～9 点までは 1 秒毎にブロックを下に落とします。そして，10～19 点のときは 0.95 秒毎，20～29 点のときは 0.9 秒毎，というように 10 点毎に 0.05 秒づつ短くしていきます。

周囲の LED は現在の点数を表します。1～9 点で 1 個，10～19 点で 2 個･･･と点灯する LED が増えていきます。点灯していない LED はブロックが落ちていくタイミングにあわせて点滅します。

ゲームオーバーで点数を表示しメロディを演奏します。なお，9 点以下と 10 点以上のときで演奏するメロディを変えます。

基本的な仕様は以上ですが，もう少しだけ。プログラムスタートでまずはゲームスタート待ちにし，周囲の LED を点滅させます。いずれかのスイッチを押したらゲームスタートです。何も押さずに 3 秒が

経過するとデモ表示を行ないます。デモ表示中でもスイッチを押すとゲームスタートします。ゲームオーバーのときはメロディの演奏が終わったらゲームスタート待ちに戻ります。

この仕様書からどんなプログラムが出来上がるのか，最初にイメージできてると理解しやすいと思います。ここで，プログラムを実際に動かしてみましょう。

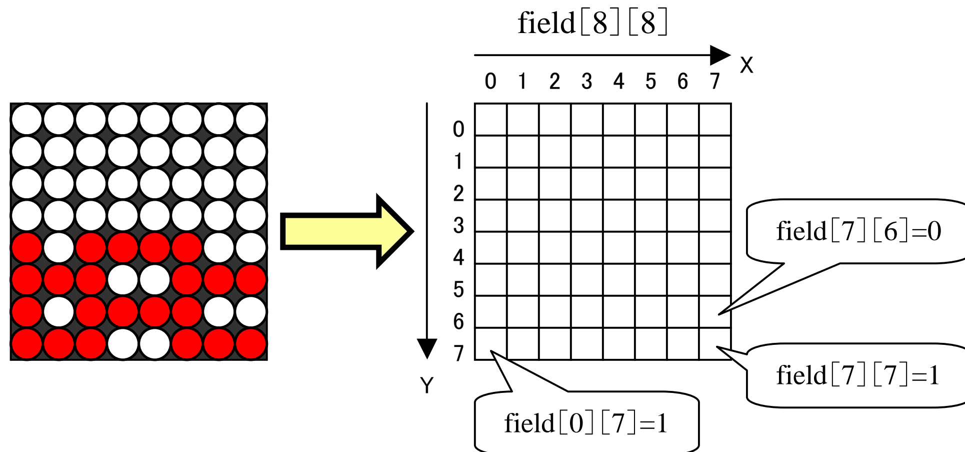
このプログラムはサイズの関係でハイパーH8でRAMにダウンロードすることはできません。それで，FDTを使ってH8/3687のフラッシュメモリにダウンロードし電源オンですぐに動くようにします。

フラッシュメモリにダウンロードするプログラムは，付属CD-R内の「**(CD-ROM)：\TK-3687mini\ｵﾌﾟｼｮﾝ\ﾀｲﾏ_LED\ﾌﾟﾛｸﾞﾗﾑ\tetris.mot**」です。FDT の使い方については CD-R 内のマニュアル，TK-3687miniは「TK-3687mini組み立て手順書」，TK-3687は「TK-3687ユーザ向け FDTでの書き込み手順」を参考にして下さい。

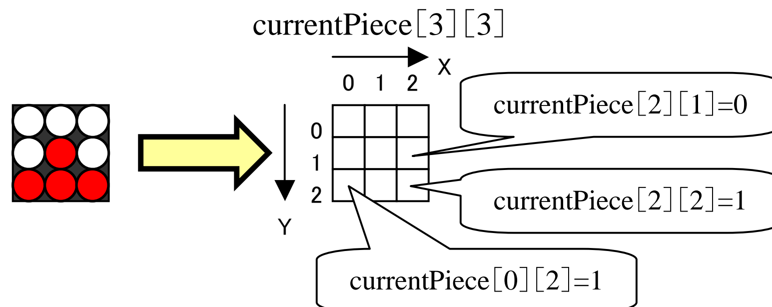
2 データの設計

仕様が決まったら，仕様どおりのプログラムを作るために，どのようなデータ構造にすればよいか検討します。ここをきちんと設計するかどうかが，すっきりしたプログラムになるか，ごちゃごちゃしたプログラムになるかの分かれ道になります。

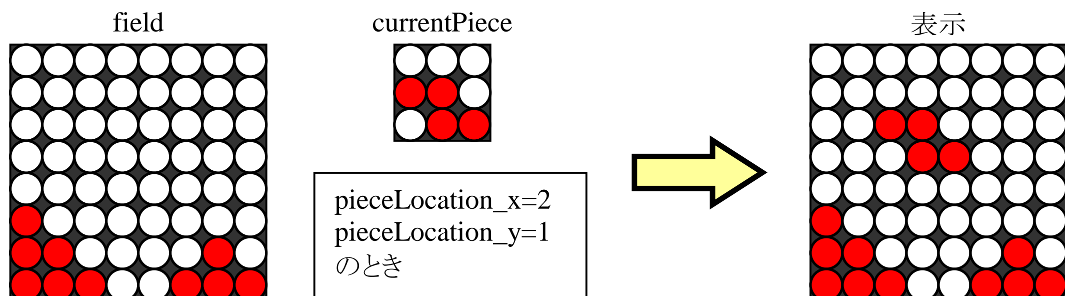
仕様書によるとゲーム画面は 8×8 ドットでした。そして，1 ドットづつ「ブロックがある/ブロックがない」という情報を持っていなければなりません。とすると，要素の数が 8×8 の二次元配列（field）を用意し，1 でブロックがある，0 でブロックがない，というように表現すればよさそうです。



次は落ちてくるブロックをどのように表現するかです。一つの案は，ゲーム画面を表す field の中に含めてしまう，という方法です。しかし，落下ブロックの移動や回転の際に，移動先のドットにすでにブロックがあるかどうかを判別していく必要があるため，複雑になりそうな予感がします。そこで，二つ目の案として，落下ブロックを表す二次元配列（currentPiece）を別に用意する方法を採用します。



currentPiece[0][0]が field のどこに位置するかを，pieceLocation_x と pieceLocation_y にセットすることにします。画面に表示するときは pieceLocation_x と pieceLocation_y をもとに field と currentPiece を重ねあわせて表示します。

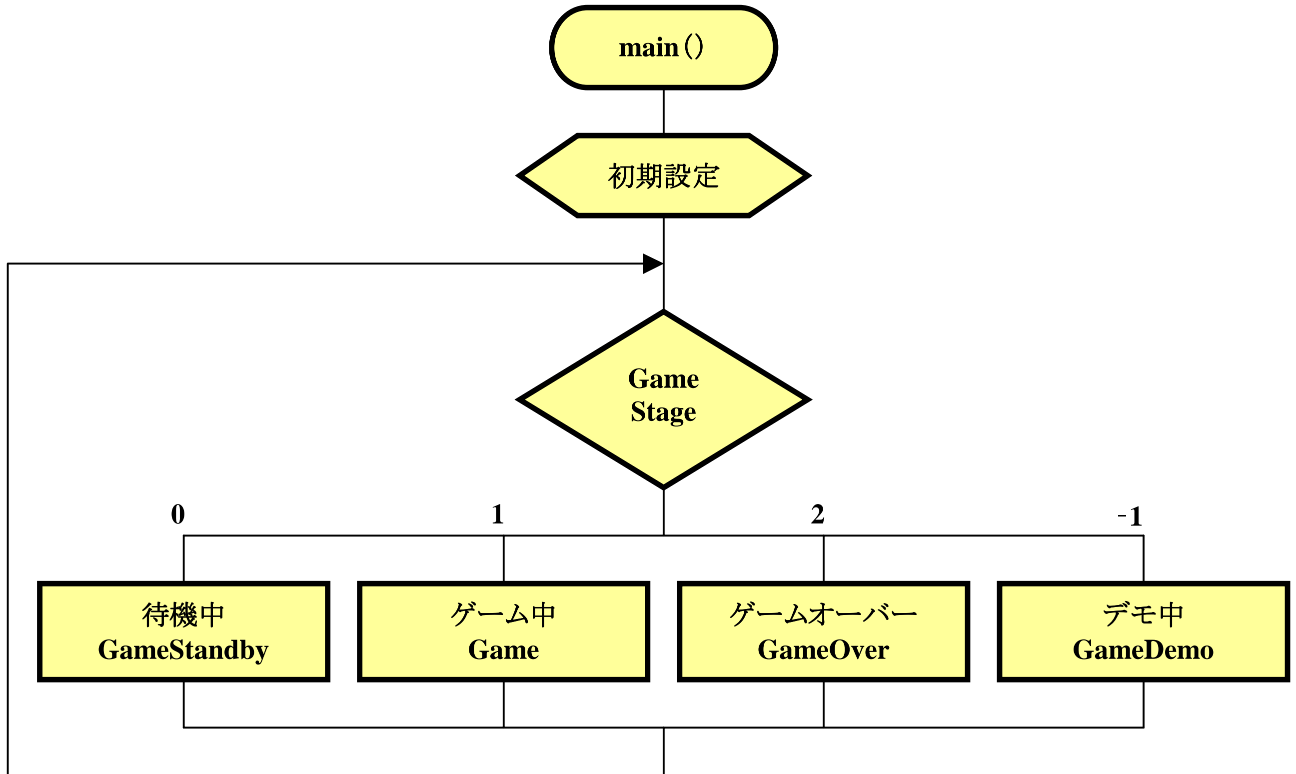


今までの説明をソースリストにまとめると次のようになります。

```
/*************************************************************************
	定数の定義（直接指定）
*************************************************************************/
#define		OK		0		//戻り値
#define		NG		-1		//戻り値
#define		TRUE		0		//戻り値
#define		FALSE		-1		//戻り値

// テトリス -------------------------------------------------------------
#define		FIELD_WIDTH	8	//ゲームフィールド(横)
#define		FIELD_HEIGHT	8	//			(縦)
#define		PIECE_LEFT	2	//ブロック左移動
#define		PIECE_RIGHT	4	//ブロック右移動
#define		PIECE_DOWN	8	//ブロック下移動
#define		MOVE_TIM_1	100	//ブロック移動間隔(100×10ms=1.00s)
#define		MOVE_TIM_2	95	//ブロック移動間隔( 95×10ms=0.95s)
#define		MOVE_TIM_3	90	//ブロック移動間隔( 90×10ms=0.90s)
#define		MOVE_TIM_4	85	//ブロック移動間隔( 85×10ms=0.85s)
#define		MOVE_TIM_5	80	//ブロック移動間隔( 80×10ms=0.80s)
#define		MOVE_TIM_6	75	//ブロック移動間隔( 75×10ms=0.75s)
#define		MOVE_TIM_7	70	//ブロック移動間隔( 70×10ms=0.70s)
#define		MOVE_TIM_8	65	//ブロック移動間隔( 65×10ms=0.65s)
#define		MOVE_TIM_9	60	//ブロック移動間隔( 60×10ms=0.60s)
#define		MOVE_TIM_10	55	//ブロック移動間隔( 55×10ms=0.55s)
#define		MOVE_TIM_11	50	//ブロック移動間隔( 50×10ms=0.50s)
#define		MOVE_TIM_12	45	//ブロック移動間隔( 45×10ms=0.45s)

/*************************************************************************
	グローバル変数の定義とイニシャライズ(RAM)
*************************************************************************/
// テトリスに関する変数 ------------------------------------------------
char		GameStage  =	0;	//ゲームステージ
					// 0:待機中
					// 1:ゲーム中
					// 2:ゲームオーバー
					//-1:デモ画面
int		Point;			//点数
char		field[FIELD_WIDTH][FIELD_HEIGHT];	//ゲームフィールド
char		currentPiece[3][3];	//現在移動中のブロック
int		pieceLocation_x;	//ブロックの位置(X座標)
int		pieceLocation_y;	//ブロックの位置(Y座標)
unsigned int	MoveTimCount;		//ブロック移動間隔カウンタ
unsigned int	MoveTimHalfCount;	//ブロック移動間隔カウンタの1/2
```

3 メインプログラム

メインプログラムでは I/O やワークエリアの初期設定のあと，GameStage によって「待機中」，「ゲーム中」，「ゲームオーバー」，「デモ中」に振り分けます。




```
/**************************************************************************
      メインプログラム
**************************************************************************/
void main(void)
{
      // イニシャライズ ---------------------------------------------------
      init_io();
      init_tmz0();
      init_tmv();
      init_tmb1();
      init_soft_timer();
      init_work_area();
      init_tetris();

      // メインループ -----------------------------------------------------
      while(1){
            switch(GameStage){
                  // 待機中 ---------------------------------------------------
                  case 0:
                        GameStandby();
                        break;

                  // ゲーム中 -------------------------------------------------
                  case 1:
                        Game();
                        break;

                  // ゲームオーバー -------------------------------------------
```

```
            case 2:
                GameOver();
                break;

            // デモ画面 ------------------------------------------------
            case -1:
                GameDemo();
                break;
        }
    }
}

/**************************************************************************
    ゲーム
**************************************************************************/
void Game(void)
{
    if      ((SwData4 & 0x08)==0x08){     //SW1が押された
        MovePiece(PIECE_RIGHT);
        SwData4 = 0;
    }
    else if ((SwData4 & 0x10)==0x10){     //SW2が押された
        MovePiece(PIECE_LEFT);
        SwData4 = 0;
    }
    else if ((SwData4 & 0x20)==0x20){     //SW3が押された
        TurnPiece();
        SwData4 = 0;
    }

    Paint();   //ゲームフィールドの表示
    setLED();  //周囲のLEDの表示
}
```

　次の章から「ゲーム中」のプログラムの内容を説明します。「待機中」，「ゲームオーバー」，「デモ中」のプログラムについてはソースリストをご覧下さい。また，LED 表示，スイッチ入力，メロディの演奏は「タイマ&LED ディスプレイ」で作成したプログラムを利用しています。これらのプログラムについて調べたい方は，CD-R 内の「タイマ&LED ディスプレイ」のマニュアルをご覧下さい。

4 ゲームアクションのプログラム

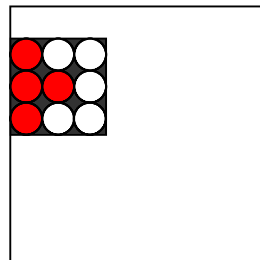
ゲームプログラムの中心はアクション部分です。「テトリス」の場合，一定時間毎にブロックを落下させたり，スイッチにあわせてブロックを左右に移動させたり回転させたり，一行そろったら削除したり，というアクションがあります。前のページのフローチャートで言えば，「ゲーム中」に行なう動作です。では，この部分を考えてみましょう。(ソースリストも併せてご覧下さい)

■ ブロックの移動（MovePiece）

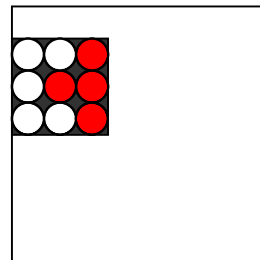
ブロックを左右に動かすことができるかどうか，ブロックを落下させることができるかどうかは，どのように判定すればよいでしょうか。ここでは左に動かすことを例に考えてみましょう。

まず，画面の大きさは決まっているので，それを越えて動くことはできません。それで，ブロックを左に動かそうとしているとき，すでに左端になっているなら動かさないようにします。ということは，pieceLocation_x=0 のときに左に動かさないということでよいでしょうか？

ところが，そう単純ではありません。下の図をご覧下さい。どちらも pieceLocation_x=0 です。左図は動かすことができません。しかし，右図はもう一列動かすことができます。つまり，右図の場合は pieceLocation_x=－1 まで動かすことができます。それで，currentPiece の最左列にブロックがあるかどうかで，判定値を使い分ける必要があります。



pieceLocation_x=0
まで動かせる



pieceLocation_x=－1
まで動かせる

もう一つ，移動できるかどうかを判別する要素は，移動先に別のブロックがあるかどうかです。もちろん，移動先にすでにブロックがあるなら移動できません。この二つの要素を判定して移動可能なときに，pieceLocation_x をマイナス 1 します。

ここでは左移動について考えてきましたが，右移動，下移動も考え方は一緒です。プログラムではブロックの移動は‘MovePiece’関数で行なっています。移動方向は関数をコールするときに引数で指定します。それで，SW1 が押されたときは右移動の引数（PIECE_LEFT）をセット，SW2 が押されたときは左移動の引数（PIECE_RIGHT）をセット，一定時間毎に下移動の引数（PIECE_DOWN）をセットして‘MovePiece’関数をコールします。

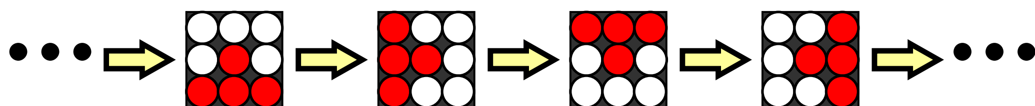
```
/***************************************************************************
     ブロックの移動判定
***************************************************************************/
int MovePiece(int move)
{
     int left,right,bottom;
     int x,y,count;

     // 左移動 ------------------------------------------------------------
     if (move==PIECE_LEFT) {
          left = GetPieceLeft();     //ブロック左側の位置情報
```

```
        if (pieceLocation_x > -(left-1)) {    //左側に移動できる
            //移動先のブロックと重なるか?
            count = 0;
            for (y=0; y<3; y++) {
                for (x=0; x<3; x++) {
                    if (((pieceLocation_x+x-1)>=0)&& ((pieceLocation_y+y)>=0)){
                        if (currentPiece[x][y] && field[pieceLocation_x + x - 1][pieceLocation_y + y]) {
                            count++;
                        }
                    }
                }
            }
            if (!count) {
                pieceLocation_x--;    //移動
            }
            else {
                return FALSE;
            }
        }
        else {
            return FALSE;
        }
    }
    // 右移動 ----------------------------------------------------------
    else if (move==PIECE_RIGHT) {
        right = GetPieceRight();   //ブロック右側の位置情報

        if (pieceLocation_x < (FIELD_WIDTH-right)) {    //右側に移動できる
            //移動先のブロックと重なるか?
            count = 0;
            for (y=0; y<3; y++) {
                for (x=0; x<3; x++) {
                    if (((pieceLocation_x+x+1)>=0)&& ((pieceLocation_y+y)>=0)){
                        if (currentPiece[x][y] && field[pieceLocation_x + x + 1][pieceLocation_y + y]) {
                            count++;
                        }
                    }
                }
            }
            if (!count) {
                pieceLocation_x++;    //移動
            }
            else {
                return FALSE;
            }
        }
        else {
            return FALSE;
        }
    }
    // 下移動 ----------------------------------------------------------
    else if (move==PIECE_DOWN) {
        bottom = GetPieceBottom(); //ブロック下側の位置情報

        if (pieceLocation_y < (FIELD_HEIGHT-bottom)) {  //下側に移動できる
            //移動先のブロックと重なるか?
            count = 0;
            for (y=0; y<3 ; y++) {
                for (x=0; x<3; x++) {
                    if (((pieceLocation_x+x)>=0)&& ((pieceLocation_y+y+1)>=0)){
                        if (currentPiece[x][y] && field[pieceLocation_x + x][pieceLocation_y + y + 1]) {
```

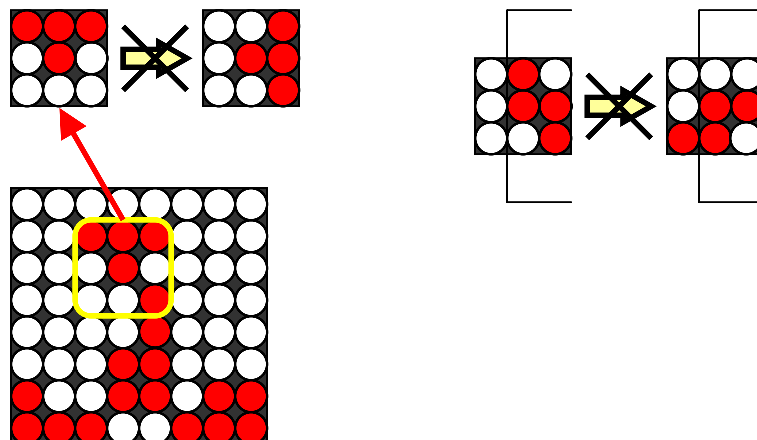
```
                                        count++;
                                    }
                                }
                            }
                        }
                        if (!count) {
                            pieceLocation_y++;    //移動
                        }
                        else {
                            return FALSE;
                        }
                    }
                    else {
                        return FALSE;
                    }
                }

                return TRUE;
}
```

■　ブロックの回転（TurnPiece）

currentPieceの回転は‘TurnPiece’関数で行ないます。SW3が押されるとコールしますが，一回コールすると時計回りに45度回転します。



ブロックの回転にもできる場合とできない場合があります。回転した結果，別のブロックと重なるようなら回転できません。また，回転した結果，ゲーム画面をはみ出すときも回転できません。



```
/****************************************************************************
        ブロックを回転させる
****************************************************************************/
int TurnPiece(void)
{
        int x,y,offsetX;
        int copy[3][3];

        //回転したブロックを生成する
        for (y=0; y<3; y++){
```

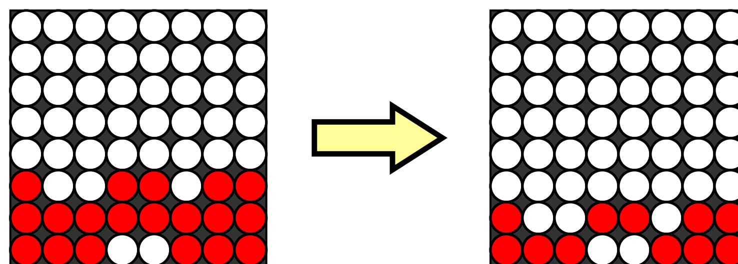
```
        for (x=0; x<3; x++){
            copy[2-y][x] = currentPiece[x][y];
        }
    }
    //回転可能かどうかを調べる
    for (y=0; y<3; y++){
        for (x=0; x<3; x++){
            if (copy[x][y]){
                offsetX = pieceLocation_x + x;
                if ((offsetX < 0) || (offsetX >= FIELD_WIDTH) || field[offsetX][pieceLocation_y + y])
                    return FALSE;
            }
        }
    }

    //copyをCurrentPieceにコピーする
    for (y=0; y<3; y++){
        for (x=0; x<3; x++){
            currentPiece[x][y] = copy[x][y];
        }
    }
    return TRUE;
}
```

■ 一定時間毎のブロック判定（TimerProc）

一定時間毎に下移動の引数（PIECE_DOWN）をセットして‘MovePiece’関数をコールします。ここで，落下ブロックが下に移動できない場合は下まで到達したと判定します。

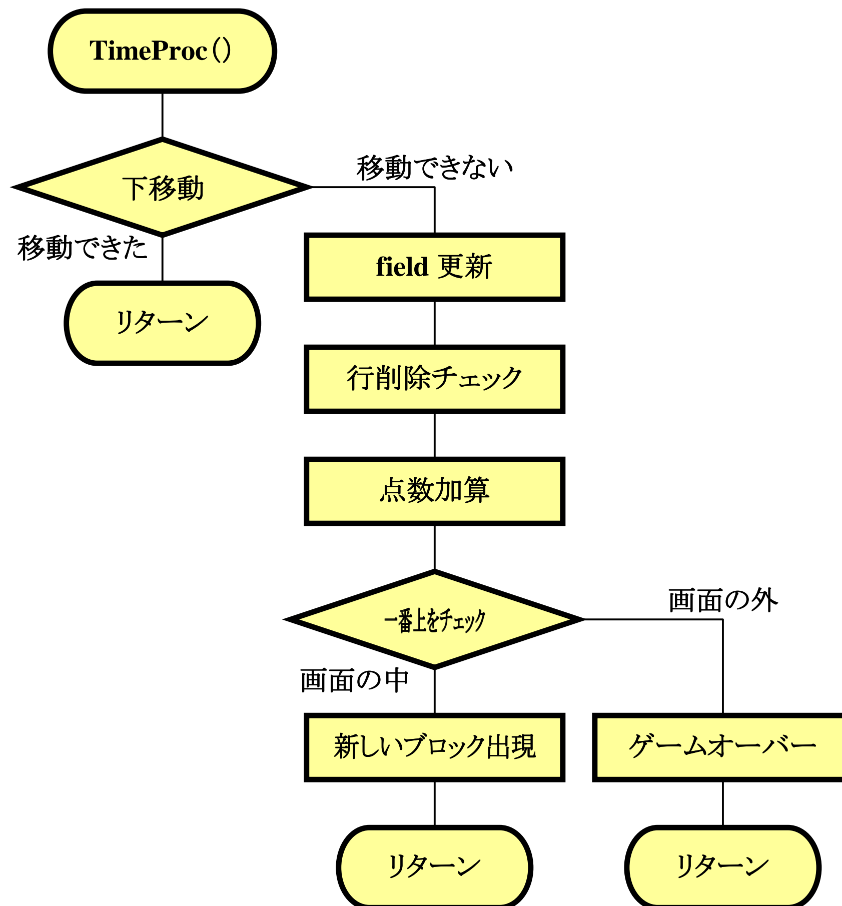
落下ブロックが下まで到達したら currentPiece を field にコピーします。

次に，一行そろっている行がないかチェックします。そろっている行は削除し，その行より上のブロックを一行下にずらします。



ここで，点数を加算します。削除した行数×行数で点数を計算します。

最後にブロックの一番上がゲーム画面の最上部を越えていないかチェックします。越えていたらゲームオーバー，まだ大丈夫なら新しいブロックを出現させます。




```
/**************************************************************************
      1秒ごとのブロック判定
**************************************************************************/
void TimerProc(void)
{
      int line, top;

      if (MovePiece(PIECE_DOWN)==FALSE){    //これより下に移動できない
            PieceToField();
            line = AdjustLine();  //行チェック，1行埋まっていたらその行は削除
            Point = Point + line * line;     //削除した行数の2乗が点数になる
            top = GetPieceTop();
            if ((pieceLocation_y + line + top - 1) < 0){
                  GameStage = 2;  //ゲームオーバー
            }
            else{
                  NextPiece();     //次のブロック
            }
      }
}
```

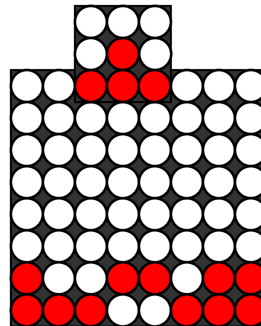
■　**新しいブロックの出現（NextPiece）**

まずは 7 種類のブロックの中から，どのブロックにするかランダムに選択します。そのために乱数を利用します。

乱数は‘rand()’関数として HEW に用意されています。‘rand()’関数は戻り値として 0～RAND_MAX の整数値を返します。次の式を使えば 0 以上 7 未満（6.9999…）の範囲の乱数を作ることができます。このうち整数部分だけ取り出して 0～6 の乱数を得ます。

$$\frac{rand()}{RAND_MAX+1}\times 7$$

currentPiece の最初の位置は，pieceLocation_x=2，pieceLocation_y=－2 にします。つまり，次のような状態で新しいブロックが出現することになります。



```
/****************************************************************************
     次のブロックへ
****************************************************************************/
void NextPiece(void)
{
     int x,y,num;

     for (y=0; y<3; y++){
          for (x=0; x<3; x++){
               currentPiece[x][y] = 0;
          }
     }

     num = (int)(((double)rand() / ((double)RAND_MAX + 1)) * 7);     //0～6の乱数を作る

     //ブロックの選択
     switch(num){
          case 0:
               currentPiece[1][0] = 1;
               currentPiece[1][1] = 1;
               currentPiece[1][2] = 1;
               break;
          case 1:
               currentPiece[1][1] = 1;
               currentPiece[2][1] = 1;
               currentPiece[1][2] = 1;
               break;
          case 2:
               currentPiece[1][1] = 1;
               currentPiece[2][1] = 1;
               currentPiece[2][2] = 1;
               break;
          case 3:
```

```
				currentPiece[0][2] = 1;
				currentPiece[1][1] = 1;
				currentPiece[1][2] = 1;
				currentPiece[2][1] = 1;
				break;
			case 4:
				currentPiece[0][1] = 1;
				currentPiece[1][1] = 1;
				currentPiece[1][2] = 1;
				currentPiece[2][2] = 1;
				break;
			case 5:
				currentPiece[0][2] = 1;
				currentPiece[1][1] = 1;
				currentPiece[1][2] = 1;
				currentPiece[2][2] = 1;
				break;
			case 6:
				currentPiece[1][1] = 1;
				currentPiece[1][2] = 1;
				currentPiece[2][1] = 1;
				currentPiece[2][2] = 1;
				break;
	}

	//位置の初期設定
	pieceLocation_x =  2;
	pieceLocation_y = -2;
}
```

■　**表示データのセット(Paint)**

ゲーム中は field と currentPiece を重ねあわせたデータを DispBuf にセットします。

```
/*************************************************************************
	表示データセット
*************************************************************************/
void Paint(void)
{
	int x,y,n;
	char d,copy_field[8][8];

	//fieldをコピー
	for (y=0; y<8 ; y++){
		for (x=0; x<8; x++){
			copy_field[x][y] = field[x][y];
		}
	}

	//currentPieceを重ねる
	for (y=0; y<3 ; y++){
		for (x=0; x<3; x++){
			if (currentPiece[x][y]){
				if (((pieceLocation_x+x)>=0)
				 && ((pieceLocation_x+x)<FIELD_WIDTH)
				 && ((pieceLocation_y+y)>=0)
				 && ((pieceLocation_y+y)<FIELD_HEIGHT)){
					copy_field[pieceLocation_x + x][pieceLocation_y + y] = currentPiece[x][y];
				}
```

```
				}
			}
		}

		//表示データに変換
		for (x=0; x<8; x++){
			d = 0;
			n = 1;
			for (y=0; y<8; y++){
				d = d + (unsigned char)(copy_field[x][y] * n);
				n = n * 2;
			}
			DispBuf[x] = d;
		}
}
```

5 プログラムを改造する

これまでのところで「テトリス」の基本的な動作はできるようになりました。十分遊べるレベルになっていると思います。それでも，ゲームをより楽しめるように工夫するところはたくさんあると思います。一例をご紹介します。

■ 落下ブロックの追加や変更

このプログラムでは 7 種類のブロックからランダムに選ぶようになっています。さらに種類を増やすことができます。ブロックの形によってはゲームの難易度がかなり高くなると思います。

■ 落下ブロックの出現箇所の変更

新しいブロックの出現箇所は固定になっています。もしこれがランダムに変更されるとしたら，先を読むことができないのでかなり難しくなるでしょうね。pieceLocation_xを 0～5 の範囲でランダムに設定すれば実現できます。最初からだと難易度が高すぎるので，ある程度点数を取ってからこのモードに入るようにしたらどうでしょうか。

■ 音の追加

ゲームに音は不可欠です。現在はゲームオーバーのときだけメロディを演奏していますが，ゲーム中 BGM を流してみてはどうでしょうか。また，行を削除するときに効果音を付けても面白いかもしれません。

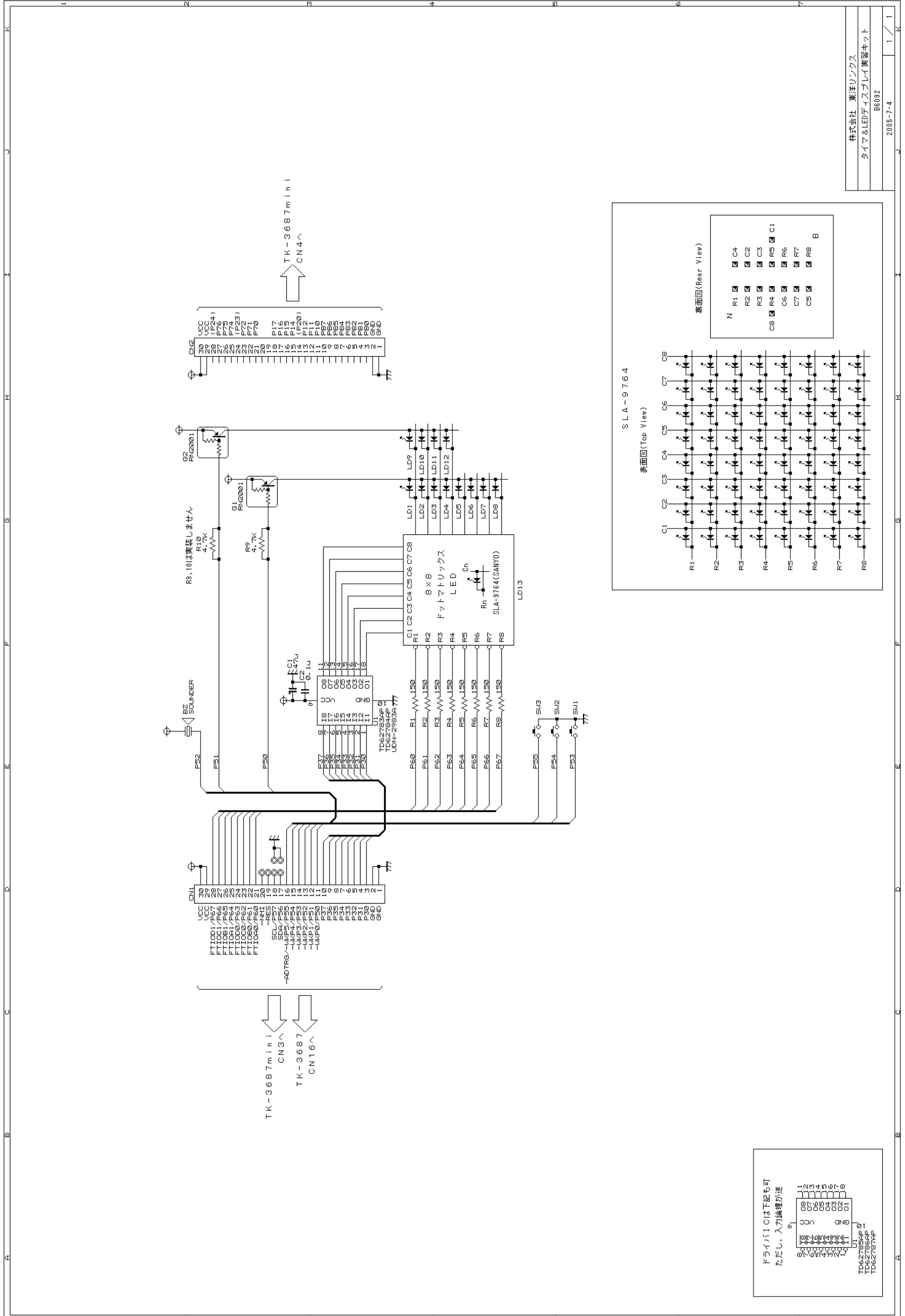
■ ルールの変更

現在はひたすら行を削除して，点数を競うようになっています。これをクリア制にしてはどうでしょうか。周囲の LED は点数を加算する毎に点灯していきますが，全て点灯したらステージクリアとします。ステージ毎に落下するブロックの形の難易度を上げたり，落下スピードを速くしたりして，段々難しくすることができるかもしれません。

自分でプログラムするということは，自分の考えた世界をマイコン上に実現できるということです。ゲームプログラムの場合，特にこの世界観というものがかなり強く反映されると思います。いろいろ工夫して自分なりの「テトリス」を作ってみてください。

付録

回路図



株式会社東洋リンクス

※ご質問はメール，または FAX で…
ユーザーサポート係（月～金 10:00～17:00，土日祝は除く）
〒102-0093　東京都千代田区平河町 1-2-2　朝日ビル
TEL：03-3234-0559
FAX：03-3234-0549
E-mail：toyolinx@va.u-netsurf.jp
URL：http://www2.u-netsurf.ne.jp/~toyolinx

20050315