1 ペントミノ

再帰手続きの例として，箱詰めパズル**ペントミノ**の解を探索します。

1.1 ペントミノとは

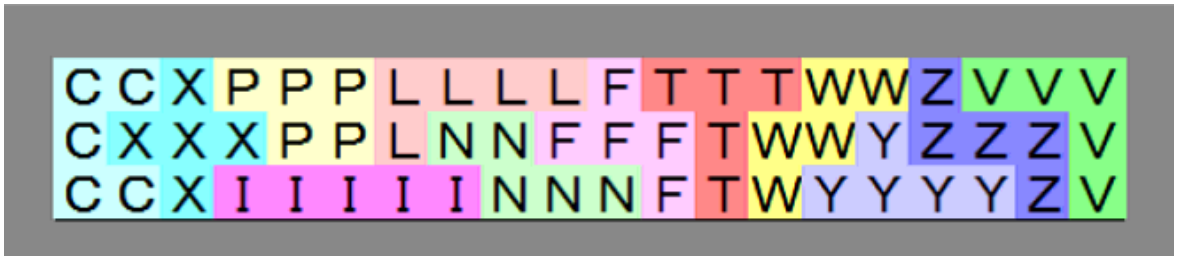
小さい正方形（ミノという）を 5 つつなげてできる形は，回転や裏返して同じになるものを同一視すると，全部で 12 個あります。これを，面積が 60（$3 \times 20, 4 \times 15, 5 \times 12, 6 \times 10$）の長方形の箱に詰め込むというパズルです。

12 個のコマには，その形に似たアルファベットで名前がついています。

L N P Y F C T V W Z I X

やや強引にこじつけたものもありますが，どんな形かわかりますか。

1.1.1 例



1.2 新規プログラム

プログラム名 Pentomino で新規プログラムを保存する。

1.2.1 Form のプロパティ

Name	FormMain
Caption	ペントミノ
Position	poDesktopCenter

1.2.2 ユニットにコメントを記入

```
unit PentominoU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls;
(************ 自分で定義する型や定数をここに書く **********)
```

```
type
  TFormMain = class(TForm)
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

var
  FormMain: TFormMain;

implementation

{$R *.DFM}
(********** 一般のモジュール **********)
(********** THako のメソッド **********)
(********** TFormMain のメソッド ******)
(********** イベントハンドラー ********)
end.
```

1.2.3 メインパネル

Panel を置く

```
Align    alBottom
Name     PanelMain
Caption
```

1.2.4 終了ボタン

Button をメインパネルの中に置く

```
Name  ButtonClose
```

OnClick イベントハンドラー

```
(********** イベントハンドラー ********)
procedure TFormMain.ButtonCloseClick(Sender: TObject);
begin
  Close;
end; {ButtonCloseClick}

end.
```

1.3 データ構造

```
(************ 自分で定義する型や定数をここに書く **********)
type
  TKomaNo = 1..12;
  TMinoNo = 1..5;
  TMukiNo = 1..8;
  TKomas  = array [TKomaNo,TMukiNo,TMinoNo] of Byte;
  TMasuNo = 0..219;
  TJouhou = 0..13;
  THako   = class(TImage)
    private
      Masus : array [TMasuNo] of TJouhou;
  end;
```

Hako はいろいろの大きさの箱ができるように，10×22 の長方形を想定しています。そして，次のようにマスニ番号 MasuNo がついていると考えます。

0	10	20	30	40	50	60	70	80	90	$\cdots$	190	200	210
1	11	21	31	41	51	61	71	81	91	$\cdots$	191	201	211
2	12	22	32	42	52	62	72	82	92	$\cdots$	192	202	212
3	13	23	33	43	53	63	73	83	93	$\cdots$	193	203	213
4	14	24	34	44	54	64	74	84	94	$\cdots$	194	204	214
5	15	25	35	45	55	65	75	85	95	$\cdots$	195	205	215
6	16	26	36	46	56	66	76	86	96	$\cdots$	196	206	216
7	17	27	37	47	57	67	77	87	97	$\cdots$	197	207	217
8	18	28	38	48	58	68	78	88	98	$\cdots$	198	208	218
9	19	29	39	49	59	69	79	89	99	$\cdots$	199	209	219

1.4 箱の生成

箱を生成して画面に描きます。

1.4.1 Hako のメソッド Create と ZenbuKaku

```
  THako   = class(TImage)
    private
      Masus : array [TMasuNo] of TJouhou;
      procedure ZenbuKaku;
    public
      constructor Create(AOwner: TComponent); override;
  end;
```

```
(********** THako のメソッド **********)
procedure THako.ZenbuKaku;
  var
```

```
    I,J: Byte;
  begin
    with Canvas do
      begin
        Brush.Color := clWhite;
        FillRect(ClipRect);
        for I := 0 to 10 do
          begin
            MoveTo(0,I*24);
            LineTo(22*24,I*24);
          end;
        for J := 0 to 22 do
          begin
            MoveTo(J*24,0);
            LineTo(J*24,10*24);
          end;
      end;
  end; {ZenbuKaku}

constructor THako.Create(AOwner: TComponent);
  begin
    inherited;
    Parent := TWinControl(AOwner);
    Width  := 24*22;
    Height := 24*10;
    Canvas.Font.Name := ' ゴシック';
    Canvas.Font.Height := 24;
    Canvas.Font.Pitch := fpFixed;
    ZenbuKaku;;
  end; {Create}
```

1.4.2 FormMain の private 部に変数を追加

private

```
    { Private 宣言 }
    Hako: THako;
```

1.4.3 Form の OnCreate ハンドラー

```
procedure TFormMain.FormCreate(Sender: TObject);
begin
  Hako := THako.Create(Self);
end; {FormCreate}
```

1.4.4 実行

左上に，10×22 の長方形が描かれます。

1.5 コマのデータ

たとえば，Ｌ型のコマを箱の中に次のように置くことを考えます。

0	10	20	30	40	50	60	70	80	90	···	190	200	210
1	11	21	31	41	51	61	71	81	91	···	191	201	211
2	12	22	32	42	52	62	72	82	92	···	192	202	212
3	13	23	33	43	53	63	73	83	93	···	193	203	213
4	14	24	34	■	54	64	74	84	94	···	194	204	214
5	15	25	35	■	55	65	75	85	95	···	195	205	215
6	16	26	36	■	56	66	76	86	96	···	196	206	216
7	17	27	37	■	■	67	77	87	97	···	197	207	217
8	18	28	38	48	58	68	78	88	98	···	198	208	218
9	19	29	39	49	59	69	79	89	99	···	199	209	219

(44,45,46,47,57) のマスが埋まります。同じ向きで，別のところに置くと，たとえば (60,61,62,63,73) のマスが埋まります。

5 つの数から先頭の数を引くと，どちらも (0,1,2,3,13) になります。この 0 から始まる 5 つの数をコマのデータとして定義しておきます。箱の中に置くときは，先頭のミノをどのマスに置くか基準になるマス（44 とか 60）を指定します。

コマによって，置く向きが（裏返しも入れると）1〜8 通りあるので，それをすべてデータとして定義しないといけません。

1.5.1 コマのデータ定義

```
type
  THako   = class(TImage)
    private
      Masus : array [TMasuNo] of TJouhou;
      procedure ZenbuKaku;
    public
      constructor Create(AOwner: TComponent); override;
  end;

const
  Name    : array [TKomaNo] of string[2]
          = ('Ｌ','Ｎ','Ｐ','Ｙ','Ｆ','Ｃ','Ｔ','Ｖ','Ｗ','Ｚ','Ｉ','Ｘ');
  MukiMax : array [TKomaNo] of TMukiNo
          = (8,8,8,8,8,4,4,4,4,4,2,1);
  Iro     : array [TKomaNo] of TColor
          = ($C0C0FF,$C0FFC0,$C0FFFF,$FFC0C0,$FFC0FF,$FFFFC0,
             $8080FF,$80FF80,$80FFFF,$FF8080,$FF80FF,$FFFF80);
  Komas : TKomas
        = (
           (( 0, 1, 2, 3,13),( 0,10,20,29,30),( 0,10,11,12,13),( 0, 1,10,20,30),
            ( 0,10, 9, 8, 7),( 0,10,20,30,31),( 0, 1, 2, 3,10),( 0, 1,11,21,31)),
           // 以下 2〜10 番のコマのデータは向きが変わってない
           // 正しいデータに書き換えなさい
           (( 0, 1, 8, 9,10),( 0, 1, 8, 9,10),( 0, 1, 8, 9,10),( 0, 1, 8, 9,10),
            ( 0, 1, 8, 9,10),( 0, 1, 8, 9,10),( 0, 1, 8, 9,10),( 0, 1, 8, 9,10)),
```

```
            (( 0, 1, 2,10,11),( 0, 1, 2,10,11),( 0, 1, 2,10,11),( 0, 1, 2,10,11),
             ( 0, 1, 2,10,11),( 0, 1, 2,10,11),( 0, 1, 2,10,11),( 0, 1, 2,10,11)),
            (( 0, 9,10,11,12),( 0, 9,10,11,12),( 0, 9,10,11,12),( 0, 9,10,11,12),
             ( 0, 9,10,11,12),( 0, 9,10,11,12),( 0, 9,10,11,12),( 0, 9,10,11,12)),
            (( 0,10,11,12,21),( 0,10,11,12,21),( 0,10,11,12,21),( 0,10,11,12,21),
             ( 0,10,11,12,21),( 0,10,11,12,21),( 0,10,11,12,21),( 0,10,11,12,21)),
            (( 0, 1, 2,10,12),( 0, 1, 2,10,12),( 0, 1, 2,10,12),( 0, 1, 2,10,12),
             ( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0)),
            (( 0,10,11,12,20),( 0,10,11,12,20),( 0,10,11,12,20),( 0,10,11,12,20),
             ( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0)),
            (( 0, 1, 2,12,22),( 0, 1, 2,12,22),( 0, 1, 2,12,22),( 0, 1, 2,12,22),
             ( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0)),
            (( 0, 1,11,12,22),( 0, 1,11,12,22),( 0, 1,11,12,22),( 0, 1,11,12,22),
             ( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0)),
            (( 0,10,11,12,22),( 0,10,11,12,22),( 0,10,11,12,22),( 0,10,11,12,22),
             ( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0)),
            (( 0, 1, 2, 3, 4),( 0,10,20,30,40),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),
             ( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0)),
            (( 0, 9,10,11,20),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),
             ( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0),( 0, 0, 0, 0, 0)));
```

データが正しいかどうか，箱の中に描いて確かめましょう。

1.5.2 Hako のメソッド KomaWoKaku

```
(********** THako のメソッド **********)
procedure THako.KomaWoKaku(Koma: TKomaNo; Muki: TMukiNo; Kijun: TMasuNo);
  var
    N : Byte;
    Masu: TMasuNo;
    I,J: Byte;
  begin
    with Canvas do
      begin
        Brush.Color := Iro[Koma];
        for N := 1 to 5 do
          begin
            Masu := Kijun+Komas[Koma,Muki][N];
            I := Masu mod 10;
            J := Masu div 10;
            TextOut(J*24,I*24,Namae[Koma]);
          end;
      end;
  end; {KomaWoKaku}
```

1.5.3 コマ指定用トラックバー

Win32/TrackBar を **2** つメインパネルの中に置く

```
Align     alLeft        alLeft
Name      TrackBarKoma  TrackBarMuki
Min       1             1
Max       12            8
Position  1             1
```

1.5.4 実行ボタン

Button をメインパネルの中に置く

```
Name  ButtonJikkou
```

OnClick ハンドラー

```
procedure TFormMain.ButtonJikkouClick(Sender: TObject);
var
  Koma: TKomaNo;
  Muki: TMukiNo;
begin
  Koma := TrackBarKoma.Position;
  Muki := TrackBarMuki.Position;
  Hako.ZenbuKaku;
  Hako.KomaWoKaku(Koma,Muki,45);
  if Muki < MukiMax[Koma]
    then TrackBarMuki.Position := Muki+1
    else begin
           TrackBarMuki.Position := 1;
           if Koma < 12
             then TrackBarKoma.Position := Koma+1
             else TrackBarKoma.Position := 1;
         end;
end; {ButtonJikkouClick}
```

1.5.5 実行

実行ボタンを押すたびに，トラックバーが自動的に変更されるので，実行ボタンを繰り返し押すだけで，すべてのデータをチェックできます。

1.6 箱を決める

1.6.1 THako に変数 Tate,Yoko を追加

```
  THako   = class(TImage)
    private
      Masus : array [TMasuNo] of TJouhou;
      No: Byte;
      Tate,Yoko: Byte;
```

1.6.2 THako のメソッド HakoWoKimeru を追加

```
procedure THako.HakoWoKimeru;
  var
    I,J: Byte;
  begin
    Tate := No+2;
    Yoko := 60 div Tate;
    with Canvas do
      begin
        Brush.Color := clGray;
        FillRect(ClipRect);
        for I := 1 to Tate+1 do
          begin
            MoveTo(24,I*24);
            LineTo((Yoko+1)*24,I*24);
          end;
        for J := 1 to Yoko+1 do
          begin
            MoveTo(J*24,24);
            LineTo(J*24,(Tate+1)*24);
          end;
      end;
    Width  := Yoko*24+48;
    Height := Tate*24+48;
    Left := (FormMain.ClientWidth-Width) div 2;
    Top  := (FormMain.ClientHeight-FormMain.PanelMain.Height-Height) div 2;
  end; {HakoWoKimeru}
```

1.6.3 箱選択ラジオグループ

RadioGroup をメインパネルの中に置く

Align	alLeft
Name	RadioGroupHako
Caption	箱の形
Items	３Ｘ２０
	４Ｘｌ５
	５Ｘｌ２
	６Ｘｌ０
Columns	4

OnClick ハンドラー

```
procedure TFormMain.RadioGroupHakoClick(Sender: TObject);
begin
  Hako.No := RadioGroupHako.ItemIndex+1;
  Hako.HakoWoKimeru;
end; {RadioGroupHakoClick}
```

1.6.4 実行

箱の形を選択すると，その形の箱（外枠が一回りある）が描かれる。

1.6.5 FormCreate で箱のデフォルトを設定

いちいち選択しなくても，実行したらすぐに箱になっていてほしい。選択（投票）しなくてもよいように，予め用意しておく値を default（棄権）と言います。

```
procedure TFormMain.FormCreate(Sender: TObject);
begin
  Hako := THako.Create(Self);
  RadioGroupHako.ItemIndex := 3; // これを追加
end; {FormCreate}
```

1.6.6 実行

最初から箱が描かれる。

1.7 コマを置く

TrackBar でコマの番号と向きを選んで， 箱の中のマスをクリックすると，そのマスを基準にしてコマを置くようにする。

1.7.1 THako のメソッド Oku を追加

```
procedure THako.Oku(Koma: TKomaNo; Muki: TMukiNo; Kijun: TMasuNo);
  var
    N : TMinoNo;
  begin
    for N := 1 to 5 do
      Masus[Kijun+Komas[Koma,Muki,N]] := Koma;
    KomaWoKaku(Koma,Muki,Kijun);
  end; {Oku}
```

1.7.2 THako のメソッド HakoMOuseDown を追加

```
procedure THako.HakoMouseDown(Sender: TObject; Button: TMouseButton;
                              Shift: TShiftState; X, Y: Integer);
  var
    I,J: Byte;
    Kijun: TMasuNo;
    Koma: TKomaNo;
    Muki: TMukiNo;
  begin
    I := Y div 24;
```

```
    J := X div 24;
    Kijun := J*10+I;
    Koma := FormMain.TrackBarKoma.Position;
    Muki := FormMain.TrackBarMuki.Position;
    if (Muki <= MukiMax[Koma])
      then Oku(Koma,Muki,Kijun);
  end; {HakoMouseDown}
```

イベントハンドラーとして登録

THako.Create の最後に 1 行追加する。

```
    ZenbuKaku;
    OnMouseDown := HakoMouseDown;
  end; {Create}
```

1.7.3 実行

まだ置けるかどうかのチェックをしていないので，重ねて置いたり，箱の外にはみ出して置いたりすることもできる。

注 1.1 右端からはみ出るように置くとエラーになる。

1.8 置けるかどうかチェックする

1.8.1 THako のメソッド Okeru を追加

```
function  THako.Okeru(Koma: TKomaNo; Muki: TMukiNo; Kijun: TMasuNo): Boolean;
  var
    N : Byte;
  begin
    N := 1;
    while (N <= 5) and (Masus[Kijun+Komas[Koma,Muki,N]] = 0) do
      Inc(N);
    Okeru := N > 5;
  end; {Okeru}
```

1.8.2 HakoMouseDown を修正

```
  //if (Muki <= MukiMax[Koma]) ↓を追加
    if (Muki <= MukiMax[Koma]) and Okeru(Koma,Muki,Kijun)
      then Oku(Koma,Muki,Kijun);
  end; {HakoMouseDown}
```

1.8.3 実行

今度は重ねておくことはできなくなった。
しかし，まだ箱の外にはみ出して置くことはできる。これを防ぐにはどうしたらよいか？

1.9 はみ出し防止

Masus[ ] の値を，箱の外側は 13，内側は 0 にすることにより，外には置けなくなります。

1.9.1 問題

HakoWoKimeru の中で上記のように設定しなさい。

1.9.2 実行

はみ出して置けなくなりましたか。

注 1.2 このパズルを楽しむためには，まだいろいろしなくてはならないことがあります。

- 選んだコマがどんな形をしているか，どういう向きかわかるように表示する。
- すでに置かれたコマを選べなくする。
- 一度置いたこまを元に戻す。

今回は，コンピュータで解を見つけることが目的なので，これらは皆さんに任せます。

1.10 探索プログラム

探索用にプログラムの名前をつけかえましょう。

名前を付けて保存	`PentominoTansakuU`
プロジェクトに名前を付けて保存	`PentominoTansakuP`

1.11 THako のメソッド KomaWoKesu，Nozoku を追加

KomaWoKaku，Oku をコピーして書き換えると楽です。

```
procedure THako.KomaWoKesu(Koma: TKomaNo; Muki: TMukiNo; Kijun: TMasuNo);
  var
    N : TMinoNo;
    Masu: TMasuNo;
    I,J: Byte;
  begin
    with Canvas do
      begin
        Brush.Color := clWhite;            // 書き換え
        for N := 1 to 5 do
```

```
          begin
            Masu := Kijun+Komas[Koma,Muki,N];
            I := Masu mod 10;
            J := Masu div 10;
            TextOut(J*24,I*24,'□');       // 書き換え
          end;
      end;
  end; {KomaWoKesu}                        //書き換え

procedure THako.Nozoku(Koma: TKomaNo; Muki: TMukiNo; Kijun: TMasuNo);
  var
    N : TMinoNo;
  begin
    for N := 1 to 5 do
      Masus[Kijun+Komas[Koma,Muki,N]] := 0; // 書き換え
    KomaWoKesu(Koma,Muki,Kijun);
  end; {Nozoku}                             // 書き換え
```

1.11.1 HakoMouseDown を変更

```
    if (Muki <= MukiMax[Koma]) and Okeru(Koma,Muki,Kijun)
      then Oku(Koma,Muki,Kijun)
      else Nozoku(Koma,Muki,Kijun);        // 追加
  end; {HakoMouseDown}
```

1.11.2 実行

11 番のブロック（Ｉ）を箱にたくさん置いてから，その上に他のブロックを置こうとすると，その形のマスが□に変わります。

1.11.3 不要なものを削除

もう TrackBar は 2 つとも要らないので削除します。
HakoMouseDown も要りません。

1.12 探索用の変数，メソッド

1.12.1 THako に変数を追加

```
  TNokori = set of TKomaNo;
  THako   = class(TImage)
    private
      Masus : array [TMasuNo] of TJouhou;
      Tate,Yoko: Byte;
      KaiNo: Integer;
      Tuzukeru: Boolean;
      Hyouji: Boolean;
```

1.12.2 THako にメソッド KaiHakken，Tansaku を追加

```
procedure THako.KaiHakken;
          {解を見つけたので，もっと続けるかどうか訊く}
  begin
    Inc(KaiNo);
    Tuzukeru := MessageDlg(IntToStr(KaiNo)+’番目の解が見つかりました’
                           +#13+#13+’もっと探しますか’,
                           mtConfirmation,[mbYes,mbNo],0) = mrYes;
  end; {KaiHakken}

procedure THako.Tansaku(Kijun: TMasuNo; Nokori: TNokori);
          {recursive procedure}
          {Kijun に Nokori に入っているコマを置いて次に進む}
  var
    Koma: TKomaNo;
    Muki: TMukiNo;
  begin
    if Tuzukeru
      then if Nokori = []
             then KaiHakken
             else begin
                    while Masus[Kijun] > 0 do
                      Inc(Kijun);
                    for Koma := 1 to 12 do
                      if Koma in Nokori
                        then for Muki := 1 to MukiMax[Koma] do
                               if Okeru(Koma,Muki,Kijun)
                                 then begin
                                        Oku(Koma,Muki,Kijun);
                                        Tansaku(Kijun+1,Nokori-[Koma]);
                                        Nozoku(Koma,Muki,Kijun);
                                      end;
                  end;
  end; {Tansaku}
```

1.12.3 実行ボタンを変更

Caption　　探索開始

OnClick ハンドラー

```
procedure TFormMain.ButtonJikkouClick(Sender: TObject);
begin
  Hako.KaiNo := 0;
  Hako.Tuzukeru := True;
  Hako.Tansaku(1,[1..12]);
end; {ButtonJikkouClick}
```

1.12.4 実行

解が見つかると表示して，探索を続けるかどうか訊くダイアログが出ます。

1.12.5 HakoWoKimeru を変更

ダイアログが解を隠してしまうので，箱の位置を変えましょう。

```
    Left := (FormMain.ClientWidth-Width) div 2;
    Top  := 48;                                    // 変更
    for I := 0 to 219 do
      Masus[I] := 13;
    for I := 1 to Tate do
      for J := 1 to Yoko do
        Masus[J*10+I] := 0;
end; {HakoWoKimeru}
```

1.13 途中経過を表示

解が見つかるまで，空の箱を見ているだけではつまらないので，途中経過を表示するようにします。

1.13.1 表示するように変更

```
    Repaint;       // 追加
end; {KomaWoKaku}
```

```
    Repaint;       // 追加
end; {KomaWoKesu}
```

1.13.2 実行

途中経過が見えます。

1.14 表示するかしないか

しかし，スピードが極端に悪化するので，表示するかしないか選べるようにしましょう。

1.14.1 表示選択 RadioGroup

RadioGroup をメインパネルの中に置く

Align	alLeft
Name	RadioGroupHyouji
Caption	途中経過を表示
Items	する
	しない
Columns	2
ItemIndex	1

1.14.2 OnClick ハンドラー

```
procedure TFormMain.RadioGroupHyoujiClick(Sender: TObject);
begin
  Hako.Hyouji := RadioGroupHyouji.ItemIndex = 0;
end; {RadioGroupHyoujiClick}
```

1.14.3 表示しないように変更

```
    if Hyouji                                          // 変更
      then Repaint;                                    // 変更
  end; {KomaWoKaku}
```

```
    if Hyouji                                          // 変更
      then Repaint;                                    // 変更
  end; {KomaWoKesu}
```

1.14.4 実行

1.15 解を書く

見つかった解を下の例のように文字列で表して Memo に書くようにします。そうするとテキストファイルに保存することもできます。

```
  N N V V V Y Y Y Y I
  F N N N V L L Y Z I
  F F F X V L Z Z Z I
  P F X X X L Z W T I
  P P C X C L W W T I
  P P C C C W W T T T
```

1.15.1 解用メモ

Memo をメインパネルの外に置く

Align	alBottom
Name	MemoKai
Font.Name	MS ゴシック
ScrollBars	ssVertical

1.15.2 Hako のメソッド KaiWoKaku を追加

```
procedure THako.KaiWoKaku;
          {解を MemoKaki に書く}
  var
    I,J: Byte;
    Gyou: String;
    Jouhou: TJouhou;
  begin
    with FormMain.MemoKai do
      begin
        Lines.Append(' 解'+IntToStr(KaiNo));
        for I := 1 to Tate do
          begin
            Gyou := '  ';
            for J := 1 to yoko do
              begin
                Jouhou := Masus[10*J+I];
                if Jouhou <= 12
                  then Gyou := Gyou+Namae[Jouhou]
                  else Gyou := Gyou+'  ';
              end;
            Lines.Append(Gyou);
          end;
        Lines.Append('');
      end;
  end; {KaiWoKaku}
```

1.15.3 Hako のメソッド KaiHakken を変更

ついでに，ダイアログも変更して，途中経過の表示/非表示をここで指定できるようにします。

```
procedure THako.KaiHakken;
          {解を見つけたので，もっと続けるかどうか訊く}
  begin
    Inc(KaiNo);
    KaiWoKaku;
    case MessageDlg(IntToStr(KaiNo)+' 番目の解が見つかりました'
                    +#13+#13+' 次の解を探索します'
                    +#13+#13+' 途中経過を表示しますか',
                    mtConfirmation,[mbYes,mbNo,mbAbort],0) of
      mrYes   : Hyouji := True;
      mrNo    : Hyouji := False;
      mrAbort : Tuzukeru := False;
    end;
  end; {KaiHakken}
```

1.15.4 ButtonJikkouClick ハンドラーに追加

```
procedure TFormMain.ButtonJikkouClick(Sender: TObject);
begin
```

```
  MemoKai.Lines.Clear;  // 追加
  Hako.KaiNo := 0;
  Hako.Tuzukeru := True;
  Hako.Tansaku(1,[1..12]);
end; {ButtonJikkouClick}
```

1.15.5 保存ボタン

Button をメインパネルの中に置く

Name	ButtonSave
Caption	解を保存

OnClick ハンドラー

```
procedure TFormMain.ButtonSaveClick(Sender: TObject);
begin
  MemoKai.Lines.SaveToFile('PentominoKai.txt');
end; {ButtonSaveClick}
```

1.15.6 実行

解がみつかったとき，ダイアログに対して

- Y を選ぶと，途中経過を表示して，探索を続ける
- N を選ぶと，途中経過を表示しないで，探索を続ける
- A を選ぶと，探索を止める

探索を止めた後で，保存ボタンを押すと，実行ファイル PentominoTansakuP.exe があるフォルダーの中に，Pentomino.txt の名前で保存される。

1.16 箱の形を追加

10.15 の例に示した 6×10 の解をよく見ると，中央に縦の分断線があり 6×5 の長方形 2 つに分けることができる。それを上下に並べると 12×5 の解にもなっている。このように，$6 \times 5 \times 2$ の形も探索できるようにしたい。

また，下の例のように，中央に 2×2 の穴が開いた 8×8 の正方形の形も興味深い。

```
  ＬＮＮＩＩＩＩＩ
  ＬＦＮＮＮＸＣＣ
  ＬＦＦＦＸＸＸＣ
  ＬＬＦ　　ＸＣＣ
  ＰＰＰ　　ＶＶＶ
  ＴＰＰＷＷＺＺＶ
  ＴＴＴＹＷＷＺＶ
  ＴＹＹＹＹＷＺＺ
```

1.16.1 RadioGroupHako を変更

```
Columns  6
Items    6X5X2  追加
         8X8-4  追加
```

1.16.2 THako に変数 No を追加

```
  TNokori = set of TKomaNo;
  THakoNo = 1..6;                                  // 追加
  THako   = class(TImage)
    private
      Masus : array [TMasuNo] of TJouhou;
      No: THakoNo;                                 // 追加
      Tate,Yoko: Byte;
```

1.16.3 RadioGroupHakoClick を変更

```
procedure TFormMain.RadioGroupHakoClick(Sender: TObject);
begin
  Hako.No := RadioGroupHako.ItemIndex+1;
  Hako.HakoWoKimeru;
end; {RadioGroupHakoClick}
```

1.16.4 HakoWoKimeru を変更

```
procedure THako.HakoWoKimeru;
  var
    I,J: Byte;
  begin
    case No of
      1..4,6 : Tate := No+2;
      else     Tate := 6;
    end;
    case No of
      1..4 : Yoko := 60 div Tate;
      6    : Yoko := 8;
      else   Yoko := 11;
    end;
    with Canvas do
    途中省略
    Top  := 48;
    for I := 0 to 219 do
      Masus[I] := 13;
    for I := 1 to Tate do
      for J := 1 to Yoko do
        Masus[J*10+I] := 0;

    case No of
      5 : begin
```

```
            // 問題　左右の 6× 5 の間に 1 列 13 を入れる
          end;
      6 : begin
            // 問題　中央の 2× 2 に 13 を入れる
          end;
  end;
end; {HakoWoKimeru}
```

1.16.5 実行

問題 $6 \times 5 \times 2$ は，1 つの解が見つかると，右の長方形を裏返したり回転した解がすぐ見つかるので，4 つずつまとまって見つかる。いずれ左の長方形を裏返したり回転した解や，左右を交換した解も見つかる。これら 32 個の解を同一視して数えると，

$$\text{本質的な解の個数} = \text{全部の解} \div 32$$

である。本質的な解はいくつあるか。