```
 if 1
GLOBAL8 equ (0x78)
TBLSIZ  equ (0x10)              ;extern const size_t TBLSIZ;// including code is
TBLADR  equ (0x00);0x20 in lin. ;extern uint8_t tbl[TBLSIZ];// responsible for..
 endif

LASTCEL equ (TBLSIZ-1)          ;const LASTCEL = TBLSIZ-1;  // TBLADR and TBLSIZ
SCOREL  equ (GLOBAL8+0)
SCOREH  equ (GLOBAL8+1)
SCOREU  equ (GLOBAL8+2)         ;uint24_t score;
POINTSL equ (GLOBAL8+3)
POINTSH equ (GLOBAL8+4)         ;uint16_t points;
TMPTILT equ (GLOBAL8+5)
TMPAPLY equ (GLOBAL8+6)
TMP8BIT equ (GLOBAL8+7)
TMP3BIT equ (PCLATH)            ;uint3_t tmp3bit; // PCLATH not used on ’313/323
                                ;                 // unless performing CALLW
DIRMASK equ (0x0f)
VALMASK equ (0xf0)
VALBITS equ (4)
NEIGHUB equ (3)
NEIGHDB equ (2)
NEIGHLB equ (1)
NEIGHRB equ (0)

NEIGHUP equ (1<<NEIGHUB)
NEIGHDN equ (1<<NEIGHDB)
NEIGHLF equ (1<<NEIGHLB)
NEIGHRT equ (1<<NEIGHRB)

OFFSTUP equ (0x100-4)
OFFSTDN equ (0+4)
OFFSTLF equ (0x100-1)
OFFSTRT equ (0+1)

flag2ofs
        btfsc   WREG,0          ;int8_t flag2ofs(uint8_t flag /*w*/){//0,1,2,4,8
        movlw   0x06            ; switch (flag) {                    //0,6,2,4,8
        lsrf    WREG,w          ; case 0:       return 0;            //0,3,1,2,4
;       andlw   0x07            ;                                      | | | | |
        brw                     ; case NEIGHLF: return -1;           //0,R,L,D,U
        retlw   0x0             ; case NEIGHDN: return +4;
        retlw   OFFSTLF         ; case NEIGHRT: return +1;
        retlw   OFFSTDN         ; case NEIGHUP: return -4;
        retlw   OFFSTRT         ; }
        retlw   OFFSTUP         ;} // flag2ofs()
;       reset                   ;
;       reset                   ;
;       reset                   ;

ofs2flag
        btfsc   WREG,NEIGHLB    ;uint4_t ofs2flag(uint8_t w, uint8_t* fsr0) {
        movlw   NEIGHLF-0x0c    ; switch (w) { case -1: return NEIGHLF & *fsr0;
        btfsc   WREG,7          ;              case -4: return NEIGHUP & *fsr0;
        addlw   0x0c            ;              default: return w & *fsr0; //RT/DN
        andwf   INDF0,w         ; } // fsr1 into fsr0, w is a flag if fsr1 valid
        return                  ;} // ofs2flag()

setfsrl
        movwf   FSR0L           ;uint4_t setfsrl(uint8_t** fsr0, uint8_t** fsr1,
        swapf   FSR0L,w         ;                 uint8_t w) { // w=fsr1flag:cell
        andlw   DIRMASK         ; *fsr0 = w;//preserve w arg before nybble swap
        btfss   STATUS,Z        ; if ((w >>= 4) != 0)         // w=0000:fsr1flag
        call    flag2ofs        ;  w = flag2ofs(w);           // w=-4,-1,0,+1,+4
        andlw   0xff            ;
        btfsc   STATUS,Z        ; if (w) { // offset FSR0L by w, store in FSR1L
        bra     fsrdone         ;  uint8_t flagarg /*FSR1H*/; // arg high nybble
        movwf   FSR1L           ;  *fsr1 = w; // FSR1 not touched if fsr1flags=0
        swapf   FSR0L,w         ;
```

```
        andlw   DIRMASK         ;
        movwf   FSR1H           ;  flagarg = *fsr0 >> 4; // returned for andwf
        movf    FSR0L,w         ;  *fsr0 &= 0x0f; // no fsr1flag nybble in FSR0L
        andlw   0x0f            ;  *fsr1 += *fsr0;
        movwf   FSR0L           ;  return flagarg;
        addwf   FSR1L,f         ; } else
        movf    FSR1H,w         ;  return 0;
fsrdone
        return                  ;} // setfsrl()

iorlf   macro   l,file          ;inline void iorlf(uint8_t* l, uint8_t* *file) {
        local i
i = 0
        while i < 8
         if (l & (1<<i))
          bsf   file,i          ; *file |= l; // STATUS, WREG unchanged
         endif
i += 1
        endw
        endm                    ;} // iorlf()

iortblf macro   tbl,fsrnum      ;inline void iortblf(uint8_t* tbl,
        local   fsrn            ;                    uint8_t* *fsrnum) {
        if (fsrnum & 3)
fsrn set 1
        else
fsrn set 0
        endif
        if tbl                  ; if (tbl && (tbl & 0x0f == 0)) // high nybble
         if (tbl & 0x0f)
 error "TBLADR not 16B aligned"
         else
          iorlf tbl,FSR#v(fsrn)L;  *fsrnum |= tbl & 0x00f0;
         endif
        else                    ; else if (tbl == 0) // bottom of linear space
         iorlf  0x20,FSR#v(fsrn)H
        endif                   ;  *fsrnum |= 0x2000;
        endm                    ;} // iortblf()

Lswitch
        brw                     ;static uint8_t initval[] = {
initval
        retlw   0x05            ;                            0x05, //  D R
        retlw   0x17            ;                            0x17, //  DLR
        retlw   0x07            ;                            0x07, //  DLR
        retlw   0x06            ;                            0x06, //  DL

        retlw   0x1d            ;                            0x1d, // UD R
        retlw   0x0f            ;                            0x0f, // UDLR
        retlw   0x0f            ;                            0x0f, // UDLR
        retlw   0x0e            ;                            0x0e, // UDL

        retlw   0x0d            ;                            0x0d, // UD R
        retlw   0x0f            ;                            0x0f, // UDLR
        retlw   0x0f            ;                            0x0f, // UDLR
        retlw   0x0e            ;                            0x0e, // UDL

        retlw   0x09            ;                            0x09, // U  R
        retlw   0x0b            ;                            0x0b, // U LR
        retlw   0x0b            ;                            0x0b, // U LR
        retlw   0x0a            ;                            0x0a};// U L
init
        movlw   LASTCEL         ;void init(void) {
        call    setfsrs         ; for (uint8_t fsr0 = &TBLADR[15];fsr0>=TBLADR;)
Linitlp
        movf    FSR0L,w         ;
        andlw   0x0f            ;
        call    Lswitch         ;
        movwi   FSR0--          ;  *fsr0-- = initval[fsr & 0x0f];
        incfsz  FSR0L,w         ;
        bra     Linitlp         ;
        return                  ;} // init()

drop
        andlw   0x0f            ;void drop(uint4_t w) {
        movwf   TMPAPLY         ; uint1_t c;
        swapf   POINTSL,w       ;
        xorwf   POINTSL,w       ; TMPAPLY = w & 0x0f; // num empties i.e. 0..14
        andlw   0x0f            ;
        movwf   TMP8BIT         ; TMP8BIT = (POINTSL^(POINTSL>>4))&0x0f; //pseudo
Lmodulo
        movf    TMPAPLY,w       ;
        subwf   TMP8BIT,w       ;
        btfsc   WREG,7          ;
        bra     Ldodrop         ;
        movwf   TMP8BIT         ;
        bra     Lmodulo         ; TMP8BIT %= TMPAPLY;
Ldodrop
        incf    TMP8BIT,w       ;
        movwf   TMP8BIT         ;
        movwf   TMPAPLY         ; TMPAPLY = TMP8BIT+=1; // for picking 2 versus 4
        movlw   LASTCEL         ;
        call    setfsrs         ; for (setfsrs(&fsr0, NULL, LASTCEL); TMP8BIT>0;)
Lnextmt
        moviw   FSR0--          ;
        andlw   VALMASK         ;
        btfss   STATUS,Z        ;
        bra     Lnextmt         ;  // TMP8BITth empty cell, address-1 into fsr0
        decfsz  TMP8BIT,f       ;  if (*fsr0-- & VALMASK == 0)
        bra     Lnextmt         ;   TMP8BIT--;
        movlw   0xaa            ;
        movwf   STATUS          ; for (w = 0xaa, c = 0; TMPAPLY > 0; TMPAPLY--)
Lpick24
        rrf     WREG,w          ;
        movwf   STATUS          ;  w = (c << 7) | (w >> 1), c = w & 1;
        decfsz  TMPAPLY,f       ;
        bra     Lpick24         ;
        andlw   0x30            ; w &= 0x30; // i.e. either (1<<0) or (1<<1), <<4
        movwf   TMPAPLY         ;
        moviw   ++FSR0          ;
        iorwf   TMPAPLY,w       ;
        movwf   INDF0           ; *++fsr0 |= w;
        return                  ;} //drop()

setfsrs
        call    setfsrl         ;uint4_t setfsrs(uint8_t** fsr0, uint8_t** fsr1,
        clrf    FSR0H           ;                uint8_t w) { // w=fsr1flag:cell
        iortblf TBLADR,FSR0     ; w = setfsrl(fsr0, fsr1, w); // w=0000:fsr1flag
        andlw   0x0f            ; *fsr0 |= TBLADR;
        btfsc   STATUS,Z        ;
        return                  ; if (w) { // requested to set fsr1 to adjacent
        andwf   INDF0,w         ;
        btfsc   STATUS,Z        ;  if (w & **fsr0 != 0)
        return                  ;   //fsr1 points to valid cell adjacent to fsr0
        clrf    FSR1H           ;   *fsr1 |= TBLADR;
        iortblf TBLADR,FSR1     ; }
        andwf   INDF0,w         ; return w & **fsr0; // STATUS Z bit=>fsr1 unset
        return                  ;} // setfsrs()

empties
        movlw   LASTCEL         ;uint5_t empties(void) {
        clrf    TMP8BIT         ; int8_t TMP8BIT = 0;
        call    setfsrs         ; for (fsr0 = &TBLADR[LASTCEL]; fsr0 >= TBLADR;)
Lnext0
        moviw   FSR0--          ;
        andlw   VALMASK         ;
        btfsc   STATUS,Z        ;  if (*fsr0-- & VALMASK == 0)
        incf    TMP8BIT,f       ;   TMP8BIT++; // found a cell with top nybble 0
```

```
        if TBLADR
         movlw  TBLADR          ;
         subwf  FSR0L,w         ;
         btfss  WREG,7          ;
        else
         incfsz FSR0L,w         ;
        endif
        bra     Lnext0          ;
        movf    TMP8BIT,w       ; return TMP8BIT;
        return                  ;} //empties()

swp2301 macro   lownybble       ;inline uint4_t swp2301(uint8_t lownybble)
        rrf     lownybble,w     ;{                                    // xxxx3210
        movwf   STATUS          ; uint1_t c = lownybble & (1<<1)/*1*/;
        rlf     lownybble,w     ; uint8_t w = (lownybble << 1) | c; // xxx32101
        andlw   0x1b            ; if ((w &= 0x1b) & (1<<4)/*3*/ == 0)
        btfsc   WREG,4          ;  return w;                          // ---32-01
        xorlw   0x14            ; else return w ^ 0x14;               // ----2301
        endm                    ;} //swp2301()

revdirs
        swp2301 TMPAPLY         ;uint4_t revdirs(uint4_t w) {return swp2301(w);}
        return                  ;//revdirs()

tiltl2r
        lslf    WREG,w          ;uint8_t tiltl2r(uint8_t w) { // iter R, bias L
        lslf    WREG,w          ; w <<= 2; // 0x2_ => 8_ if L, 0x1_ => 0x40 if R
        xorlw   0x50            ; return w ^ 0x50; // 0xd_ if L, 0x1_ if R
        return                  ;} //tiltl2r()

tiltr2l
        call    tiltl2r         ;uint8_t tiltr2l(uint8_t w) { // iter L, bias R
        xorlw   0xf3            ; return tiltl2r(w) ^ 0xf0 ^ 0x03;
        return                  ;} //tiltr2l()

tltstrt
        andlw   0xf3            ;uint8_t tltstrt(uint8_t w) {  // w=UDLR00nn
        btfsc   WREG,7          ; switch(w >> 4) { // (iteration dir,=-bias dir)
        addlw   0x0c            ; case NEIGHUP: return w + 12;    //0x8c,d,e,f
        btfsc   WREG,5          ; case NEIGHLF: return tiltr2l(w);//0x23,7,b,f
        bra     tiltr2l         ; case NEIGHRT: return tiltl2r(w);//0x10,4,8,c
        btfsc   WREG,4          ; case NEIGHDN: default: return w;//0x40,1,2,3
        bra     tiltl2r         ; }
        return                  ;} //tltstrt()

tltpair
        movlw   VALMASK         ;int8_t tltpair(uint8_t* fsr0, uint8_t* fsr1){
        andwf   INDF0,w         ; // nonzero else swap adjacent nonzero else -1
        btfss   STATUS,Z        ; if (*fsr0 & VALMASK)
        retlw   0               ;  return 0;
        movlw   VALMASK         ; else if (*fsr1 & VALMASK) {
        andwf   INDF1,w         ;  // *fsr0=0000UDLR but swap *fsr0 and *fsr1
        btfsc   STATUS,Z        ;  *fsr0 = (*fsr1 & VALMASK)|(*fsr0 & DIRMASK);
        retlw   0xff            ;  *fsr1 = 0                |(*fsr1 & DIRMASK);
        iorwf   INDF0,f         ;  return 1;
        movlw   DIRMASK         ; } else  // *fsr1=0000UDLR also, keep looking
        andwf   INDF1,f         ;  return -1;
        retlw   1               ;} //tltpair()

nxtpair
        andlw   DIRMASK         ;uint4_t nxtpair(uint8_t* *fsr0,uint8_t* *fsr1,
        btfsc   STATUS,Z        ;                uint4_t w) { // w=0 is fastest
        bra     Ldirunk         ; if (w & DIRMASK)
        andwf   INDF0,w         ;
        call    flag2ofs        ;  w = flag2ofs(w & **fsr0); // -4,-1,+1 or +4
        bra     Lnxtadd         ;
Ldirunk
        movf    FSR0L,w         ; else
        subwf   FSR1L,w         ;  w = *fsr1 - *fsr0; //-4, -1, +1 or +4
```

```
Lnxtadd
        addwf   FSR0L,f         ; *fsr0 += w;
        addwf   FSR1L,f         ; *fsr1 += w;
        call    ofs2flag        ;
        andwf   INDF0,w         ; return ofs2flag(w) & **fsr0;// Z=>FSR1 invalid
        return                  ;} // nxtpair()

savefsr
        movf    FSR0L,w         ;void savefsr(uint8_t* fsr0, uint8_t* fsr1) {
        subwf   FSR1L,w         ;
        call    ofs2flag        ; w = ofs2flag(fsr1 - fsr0);
;       andwf   INDF0,w         ;
        andlw   DIRMASK         ;
        movwf   TMP8BIT         ;
        swapf   FSR0L,w         ; *TMP8BIT = ((fsr0&0x0f)<<4)|(*fsr0&w&DIRMASK);
        if TBLADR
         andlw  0xf0            ;
        endif
        iorwf   TMP8BIT,f       ; return (fsr0 & 0x0f) << 4;
        return                  ;} //savefsr()

tltmult
        call    savefsr         ;uint1_t tltmult(uint8_t* fsr0, uint8_t* fsr1,
Lfsr1n0
        movlw   DIRMASK         ;                 uint8_t w) {
        andwf   TMP8BIT,w       ; savefsr(fsr0, fsr1); // TMP8BIT=cel#:UDLR
        call    nxtpair         ; while(1)
        btfsc   STATUS,Z        ;  if (nxtpair(&fsr0,&fsr1,TMP8BIT&DIRMASK)==0)
        bra     Lallzer         ;   break; // past end of row/column
        movlw   VALMASK         ;
        andwf   INDF1,w         ;
        btfsc   STATUS,Z        ;
        bra     Lfsr1n0         ;  else if (*fsr1 & VALMASK) {// found nonzero
        swapf   TMP8BIT,w       ;
        andlw   0x0f            ;   // restore fsr0 only, move *fsr1 into it
        call    setfsrs         ;   setfsrs(&fsr0, NULL, TMP8BIT >> 4);
        movlw   VALMASK         ;
        andwf   INDF1,w         ;
        iorwf   INDF0,f         ;   *fsr0 |= *fsr1 & VALMASK;
        movlw   DIRMASK         ;   *fsr1 &= DIRMASK;
        andwf   INDF1,f         ;   w = (TMP8BIT<<4)|(TMP8BIT>>4);
        swapf   TMP8BIT,w       ;   setfsrs(&fsr0, &fsr1, w);
        call    setfsrs         ;   return 1; // scoot flag for POINTS;
        retlw   1               ;  }
Lallzer
;       swapf   TMP8BIT,w       ; setfsrs(&fsr0,&fsr1,(TMP8BIT<<4)|(TMP8BIT>>4));
;       call    setfsrs         ; return 0; // indicates done with this row/col
        retlw   0               ;} // tltmult()

tilt
        movwf   TMPAPLY         ;uint4_t tilt(uint4_t w) { // w=DURL, preserved
Lnxttlt
        swapf   TMPAPLY,w       ; for (TMPAPLY=w; TMPAPLY<0x40; TMPAPLY+=0x10) {
        call    tltstrt         ;  uint8_t w = (TMPAPLY << 4) | (TMPAPLY >> 4);
        call    setfsrs         ;  uint1_t z = 0;
Lnxtcel
        call    tltpair         ;  for (setfsrs(&fsr0, &fsr1, tltstrt(w&0xf3));
        btfss   WREG,7          ;       !z; z = (nxtpair(&fsr0, &fsr1, 0)==0)) {
        bra     Laddpnt         ;   int8_t w = tltpair(fsr0, fsr1);//bias to dir
                                ;   if (w < 0) {// both *fsr0 and *fsr1 are zero
        call    tltmult         ;    w = tltmult(fsr0, fsr1); // look past fsr1
        btfss   WREG,0          ;    if (w == 0)
        bra     Ltltcnt         ;     break;// if the rest of row/col also zero,
Laddpnt
        iorwf   POINTSL,f       ;
        if 0
         movlw  DIRMASK         ;
         andwf  TMPAPLY,w       ;
        else
         clrw                   ;
        endif
        call    nxtpair         ;   }//... break inner loop, skip nxtpair()
        btfss   STATUS,Z        ;   POINTSL |= w;
        bra     Lnxtcel         ;  }
Ltltcnt
        movlw   0x10            ;
        addwf   TMPAPLY,f       ;
        btfss   TMPAPLY,6       ;
        bra     Lnxttlt         ; }
        movf    TMPAPLY,w       ;
        andlw   DIRMASK         ; return TMPAPLY;     // w=0000DURL
        return                  ;} // tilt()

lutpow2
        andlw   0x07            ;uint8_t lutpow2(uint3_t w) { switch(w & 0x07) {
        brw                     ; case 0: retlw 0x01;
        retlw   0x01            ; case 1: retlw 0x02;
        retlw   0x02            ; case 2: retlw 0x04;
        retlw   0x04            ; case 3: retlw 0x08;
        retlw   0x08            ; case 4: retlw 0x10;
        retlw   0x10            ; case 5: retlw 0x20;
        retlw   0x20            ; case 6: retlw 0x40;
        retlw   0x40            ; case 7: retlw 0x80;
        retlw   0x80            ;} } // lutpow2(), 8-bit result

p2m16   macro   file            ;inline uint8_t p2m16(uint4_t w, uint8_t *file){
        movwf   file            ; *file = w;
        call    lutpow2         ; w = lutpow2(w);
        btfss   file,3          ;
        bra     $+3             ; if (*file & 0x08)
        movwf   file            ;  { *file = w; return 0; } // w into high byte
        retlw   0x00            ; else
        clrf    file            ;  { *file = 0; return w; } // w into low byte
        return                  ;
        endm                    ;} // p2m16(), 16-bit result
pow2wf
        p2m16   TMP8BIT        ;uint8_t pow2wf(int w){return p2m16(w,&TMP8BIT);}

cpspair
;       movlw   VALMASK         ;//by caller
;       andwf   INDF1,w         ;
        xorwf   INDF0,w         ;int8_t cpspair(uint8_t* fsr0, uint8_t* fsr1){
        andlw   VALMASK         ; //
        btfss   STATUS,Z        ; if (*fsr1 & VALMASK != *fsr0 & VALMASK)
        retlw   0               ;  return 0; // no points earned this step

        movlw   1<<VALBITS      ;
        addwf   INDF0,f         ; *fsr0 += 1<<VALBITS;
;       btfsc   STATUS,C        ;
;       reset                   ; // 32768+32768>65535 not even possible?
        call    savefsr         ; savefsr(fsr0, fsr1); // TMP8BIT=cel#:UDLR
Lscoot
        clrw                    ;
        call    nxtpair         ; while (nxtpair(&fsr0, &fsr1, 0)) {
        btfsc   STATUS,Z        ;
        bra     Lzerend         ;
        movlw   DIRMASK         ;
        andwf   INDF0,f         ;
        movlw   VALMASK         ;
        andwf   INDF1,w         ;
        iorwf   INDF0,f         ;  *fsr0 = (*fsr1 & VALMASK) | (*fsr0 & DIRMASK);
        movlw   DIRMASK         ;
        andwf   INDF1,f         ;  *fsr1 =                 0 | (*fsr1 & DIRMASK);
        bra     Lscoot          ; }
Lzerend
        movlw   DIRMASK         ;
        andwf   INDF0,f         ; *fsr0 = 0 | (*fsr0 & DIRMASK);
        swapf   TMP8BIT,w       ;
        call    setfsrs         ; setfsrs(&fsr0, &fsr1, (TMP8BIT<<4)|(TMP8BIT>>4));
        swapf   INDF0,w         ;
        andlw   VALMASK         ; return *fsr0 >> 4;
        return                  ;} // cpspair()

collaps
        movwf   TMPAPLY         ;uint4_t collaps(uint4_t w) { // w=DURL, preserved?
Lnxtcps
        swapf   TMPAPLY,w       ; for (TMPAPLY=w; TMPAPLY<0x40; TMPAPLY+=0x40) {
        call    tltstrt         ;  uint8_t w = (TMPAPLY << 4) | (TMPAPLY >> 4);
        call    setfsrs         ;  uint1_t z = 0;
Lnxtcec
        movf    INDF1,w         ;  for (setfsrs(&fsr0, &fsr1, tltstrt(w&0xf3));
        andlw   VALMASK         ;       !z; z = (nxtpair(&fsr0, &fsr1, 0)==0)) {
        btfsc   STATUS,Z        ;
        bra     Lcpscnt         ;   int8_t w;
        call    cpspair         ;   if (*fsr1 == 0) break; //done with row/column
        andlw   0x0f            ;   w = cpspair(fsr0, fsr1);//else try to combine
        btfsc   STATUS,Z        ;   if (w & 0x0f) { // 0=NA, 2=got 4, 3=got 8
        bra     Lnopnt          ;    uint8_t TMP8BIT = ((1<<w) & 0xff00) >> 8;
        call    pow2wf          ;                  w =  (1<<w) & 0x00ff;
;       btfsc   WREG,1          ;    if (w == 2)
;       bra     Lcpscnt         ;     break; // already handled above
        addwf   POINTSL,f       ;
        movf    TMP8BIT,w       ;    *((uint16_t*) &POINTSL) += (TMP8BIT << 8)|w;
        addwfc  POINTSH,f       ;   }
Lnopnt
        if 0
         movlw  DIRMASK         ;
         andwf  TMPAPLY,w       ;
        else
         clrw                   ;
        endif
        call    nxtpair         ;
        btfss   STATUS,Z        ;
        bra     Lnxtcec         ;  }
Lcpscnt
        movlw   0x10            ;
        addwf   TMPAPLY,f       ;
        btfss   TMPAPLY,6       ;
        bra     Lnxtcps         ; }
        movf    POINTSL,w       ;
        iorwf   POINTSH,w       ; return (POINTSH >> 8) | POINTSL; // 0=bad move
        return                  ;} // collaps()

apply
        clrf    POINTSL         ;uint8_t apply(uint4_t w) { // w=0000UDLR
        clrf    POINTSH         ; POINTS = 0; // 1 for tilt, >1 for collapse
        movwf   TMPAPLY         ;
        call    revdirs         ; w = revdirs(TMPAPLY = w);// iterate oppositely
        call    tilt            ; tilt(w);
        call    collaps         ; return collaps(w);
        return                  ;} // apply()

turn
        andlw   DIRMASK         ;uint8_t turn(uint4_t w) { // w=0000UDLR
        btfsc   STATUS,Z        ; if (w & DIRMASK == 0)
        retlw   0               ;  return 0; // no move given
        call    apply           ;
        btfsc   STATUS,Z        ; if (apply(w) == 0)
        retlw   0               ;  return 0; // bad move: no slide/combine
        movlw   0xfe            ;
        andwf   POINTSL,w       ; POINTSL &= 0xfe; // not really 1 pt for slide
        addwf   SCOREL,f        ;
        movf    POINTSH,w       ;
        addwfc  SCOREH,f        ;
        clrw                    ;
        addwfc  SCOREU,f        ; *((uint24_t*)&SCOREL)+=*((uint16_t*)&POINTSL);
        call    empties         ;
        call    drop            ; drop(empties());
        movlw   VALMASK         ;
        andwf   INDF0,w         ;
        xorwf   FSR0L,w         ; return (*fsr0 & VALMASK) ^ (fsr0 & 0x00ff);
        return                  ;} // turn()

newgame
        clrf    SCOREL          ;void newgame(void) {
        clrf    SCOREH          ; SCOREU = SCOREH = SCOREL = 0;
        clrf    SCOREU          ;
        call    init            ; init();
        return                  ;}
```