15 puzzle game

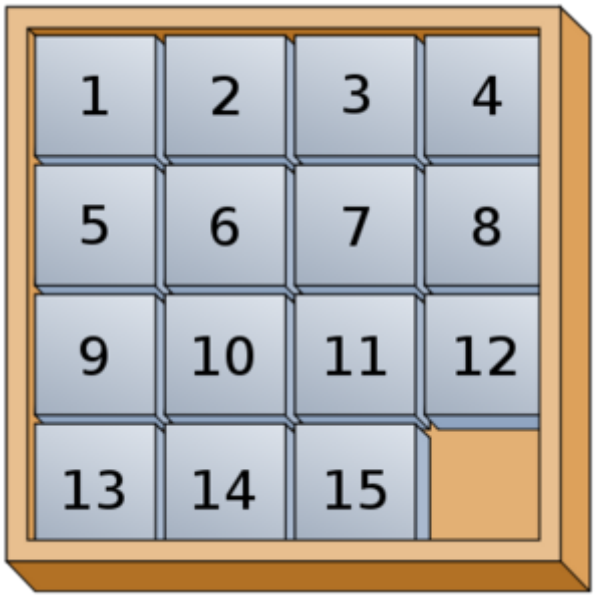
15 puzzle game
You are encouraged to solve this task according to the task description, using any language you may know.

Task

Implement the Fifteen Puzzle Game.

The **15-puzzle** is also known as:

- **Fifteen Puzzle**
- **Gem Puzzle**
- **Boss Puzzle**
- **Game of Fifteen**
- **Mystic Square**
- **14-15 Puzzle**
- and some others.



Related Tasks

- 15 Puzzle Solver
- 16 Puzzle Game

Contents

11l
68000 Assembly
AArch64 Assembly
Action!
Ada
Amazing Hopper
APL
ARM Assembly
Arturo
Astro
AutoHotkey
BASIC
BBC BASIC

BQN
C
C#
C++
COBOL
Common Lisp
EasyLang
F#
Factor
Forth
Fortran
FreeBASIC
Gambas
Go
Harbour
Haskell
J
Java
JavaScript
Julia
Kotlin
Liberty BASIC
LiveCode
Lua
M2000 Interpreter
Mathematica / Wolfram Language
Mercury
MUMPS
Nim
OCaml
Pascal
Perl
Phix
PHP
Picat
Powershell
Processing
PureBasic
Python
QB64

Quackery
R
Racket
Raku
Rebol
Red
REXX
Ring
Ruby
Run BASIC
Rust
Scala
Scheme
Scilab
Simula
Standard ML
Tcl
UNIX Shell
VBA
Visual Basic .NET
Visual Prolog
VBScript
Wren
x86-64 Assembly
XBasic
XPL0
Yabasic

11l

Translation of: Python: Original, with output

```
T Puzzle
   position = 0
   [Int = String] items

   F main_frame()
      V& d = .items
      print('+-----+-----+-----+-----+')
      print('|#.|#.|#.|#.|'.format(d[1], d[2], d[3], d[4]))
      print('+-----+-----+-----+-----+')
      print('|#.|#.|#.|#.|'.format(d[5], d[6], d[7], d[8]))
      print('+-----+-----+-----+-----+')
      print('|#.|#.|#.|#.|'.format(d[9], d[10], d[11], d[12]))
      print('+-----+-----+-----+-----+')
      print('|#.|#.|#.|#.|'.format(d[13], d[14], d[15], d[16]))
      print('+-----+-----+-----+-----+')

   F format(=ch)
      ch = ch.trim(' ')
```

```
      I ch.len == 1
         R ‘  ’ch‘  ’
      E I ch.len == 2
         R ‘  ’ch‘ ’
      E
         assert(ch.empty)
         R ‘     ’

   F change(=to)
      V fro = .position
      L(a, b) .items
         I b == .format(String(to))
            to = a
            L.break
      swap(&.items[fro], &.items[to])
      .position = to

   F build_board(difficulty)
      L(i) 1..16
         .items[i] = .format(String(i))
      V tmp = 0
      L(a, b) .items
         I b == ‘  16 ’
            .items[a] = ‘     ’
            tmp = a
            L.break
      .position = tmp
      Int diff
      I difficulty == 0
         diff = 10
      E I difficulty == 1
         diff = 50
      E
         diff = 100
      L 0 .< diff
         V lst = .valid_moves()
         [Int] lst1
         L(j) lst
            lst1.append(Int(j.trim(‘ ’)))
         .change(lst1[random:(lst1.len)])

   F valid_moves()
      V pos = .position
      I pos C [6, 7, 10, 11]
         R [.items[pos - 4], .items[pos - 1], .items[pos + 1], .items[pos + 4]]
      E I pos C [5, 9]
         R [.items[pos - 4], .items[pos + 4], .items[pos + 1]]
      E I pos C [8, 12]
         R [.items[pos - 4], .items[pos + 4], .items[pos - 1]]
      E I pos C [2, 3]
         R [.items[pos - 1], .items[pos + 1], .items[pos + 4]]
      E I pos C [14, 15]
         R [.items[pos - 1], .items[pos + 1], .items[pos - 4]]
      E I pos == 1
         R [.items[pos + 1], .items[pos + 4]]
      E I pos == 4
         R [.items[pos - 1], .items[pos + 4]]
      E I pos == 13
         R [.items[pos + 1], .items[pos - 4]]
      E
         assert(pos == 16)
         R [.items[pos - 1], .items[pos - 4]]

   F game_over()
      V flag = 0B
      L(a, b) .items
         I b != ‘     ’
            I a == Int(b.trim(‘ ’))
               flag = 1B
            E
               flag = 0B
      R flag

V g = Puzzle()
g.build_board(Int(input("Enter the difficulty : 0 1 2\n2 => highest 0 => lowest\n")))
g.main_frame()
print(‘Enter 0 to exit’)
L
```

```
   print("Hello user:\nTo change the position just enter the no. near it")
   V lst = g.valid_moves()
   [Int] lst1
   L(i) lst
      lst1.append(Int(i.trim(‘ ’)))
      print(i.trim(‘ ’)" \t", end' ‘’)
   print()
   V x = Int(input())
   I x == 0
      L.break
   E I x !C lst1
      print(‘Wrong move’)
   E
      g.change(x)
   g.main_frame()
   I g.game_over()
      print(‘You WON’)
      L.break
```

Output:

The same as in Python.

68000 Assembly

This is an entire Sega Genesis game, tested in the Fusion emulator. Thanks to Keith S. of Chibiakumas for the cartridge header, font routines, and printing logic. I programmed the actual game logic. This code can be copied and pasted into a text file and assembled as-is using vasmm68k_mot_win32.exe, no includes or incbins necessary (even the bitmap font is here too.)

```
;15 PUZZLE GAME
;Ram Variables
Cursor_X equ $00FF0000      ;Ram for Cursor Xpos
Cursor_Y equ $00FF0000+1    ;Ram for Cursor Ypos
joypad1 equ $00FF0002

GameRam equ $00FF1000       ;Ram for where the pieces are
GameRam_End equ $00FF100F   ;the last valid slot in the array
;Video Ports
VDP_data    EQU $C00000 ; VDP data, R/W word or longword access only
VDP_ctrl    EQU $C00004 ; VDP control, word or longword writes only

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                   VECTOR TABLE
;org $00000000
    DC.L    $00FFFFFE       ;SP register value
    DC.L    ProgramStart    ;Start of Program Code
    DC.L    IntReturn       ; bus err
    DC.L    IntReturn       ; addr err
    DC.L    IntReturn       ; illegal inst
    DC.L    IntReturn       ; divzero
    DC.L    IntReturn       ; CHK
    DC.L    IntReturn       ; TRAPV
    DC.L    IntReturn       ; privilege viol
    DC.L    IntReturn       ; TRACE
    DC.L    IntReturn       ; Line A (1010) emulator
    DC.L    IntReturn       ; Line F (1111) emulator
    DC.L    IntReturn,IntReturn,IntReturn,IntReturn     ; Reserved /Coprocessor/Format err/ Uninit Interrupt
    DC.L    IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn
    DC.L    IntReturn       ; spurious interrupt
    DC.L    IntReturn       ; IRQ level 1
    DC.L    IntReturn       ; IRQ level 2 EXT
    DC.L    IntReturn       ; IRQ level 3
    DC.L    IntReturn       ; IRQ level 4 Hsync
    DC.L    IntReturn       ; IRQ level 5
    DC.L    IntReturn       ; IRQ level 6 Vsync
    DC.L    IntReturn       ; IRQ level 7 (NMI)
;org $00000080
```

```
;TRAPS
    DC.L    IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn
    DC.L    IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn
;org $000000C0
;FP/MMU
    DC.L    IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn
    DC.L    IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn,IntReturn



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                   Header
HEADER:
    DC.B    "SEGA GENESIS    "          ;System Name    MUST TAKE UP 16 BYTES, USE PADDING IF NECESSARY
    DC.B    "(C)PDS  "          ;Copyright  MUST TAKE UP 8 BYTES, USE PADDING IF NECESSARY
    DC.B    "2022.JUN"          ;Date       MUST TAKE UP 8 BYTES, USE PADDING IF NECESSARY
CARTNAME:
    DC.B    "15 PUZZLE"
CARTNAME_END:
    DS.B 48-(CARTNAME_END-CARTNAME) ;ENSURES PROPER SPACING
CARTNAMEALT:
    DC.B    "15 PUZZLE"
CARTNAMEALT_END:
    DS.B 48-(CARTNAMEALT_END-CARTNAMEALT)   ;ENSURES PROPER SPACING
gameID:
    DC.B    "GM PUPPY001-00"            ;TT NNNNNNNN-RR T=Type (GM=Game) N=game Num  R=Revision
    DC.W    $0000               ;16-bit Checksum (Address $000200+)
CTRLDATA:
    DC.B    "J               "          ;Control Data (J=3button K=Keyboard 6=6button C=cdrom)
                                        ;(MUST TAKE UP 16 BYTES, USE PADDING IF NECESSARY)
ROMSTART:
    DC.L    $00000000           ;ROM Start
ROMLEN:
    DC.L    $003FFFFF           ;ROM Length
RAMSTART:
    DC.L    $00FF0000
RAMEND:
    DC.L    $00FFFFFF   ;RAM start/end (fixed)

    DC.B    "            "      ;External RAM Data  (MUST TAKE UP 12 BYTES, USE PADDING IF NECESSARY)
    DC.B    "            "      ;Modem Data     (MUST TAKE UP 12 BYTES, USE PADDING IF NECESSARY)
MEMO:
    DC.B    "                                        "      ;(MUST TAKE UP 40 BYTES, USE PADDING IF NECESSARY)
REGION:
    DC.B    "JUE             "  ;Regions Allowed        (MUST TAKE UP 16 BYTES, USE PADDING IF NECESSARY)
    even
HEADER_END:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                   Generic Interrupt Handler
IntReturn:
    rte                             ;immediately return to game
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                   Program Start
ProgramStart:
    ;initialize TMSS (TradeMark Security System)
    move.b ($A10001),D0     ;A10001 test the hardware version
    and.b #$0F,D0
    beq NoTmss              ;branch if no TMSS chip
    move.l #'SEGA',($A14000);A14000 disable TMSS
NoTmss:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                   Set Up Graphics

    lea VDPSettings,A5            ;Initialize Screen Registers
    move.l #VDPSettingsEnd-VDPSettings,D1 ;length of Settings

    move.w (VDP_ctrl),D0                ;C00004 read VDP status (interrupt acknowledge?)
    move.l #$00008000,d5                ;VDP Reg command (%8rvv)

NextInitByte:
    move.b (A5)+,D5               ;get next video control byte
    move.w D5,(VDP_ctrl)                ;C00004 send write register command to VDP
        ;   8RVV - R=Reg V=Value
    add.w #$0100,D5               ;point to next VDP register
    dbra D1,NextInitByte                ;loop for rest of block

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;       Set up palette

    ;Define palette
    move.l #$C0000000,d0    ;Color 0 (background)
    move.l d0,VDP_Ctrl
    ;        ----BBB-GGG-RRR-
    move.w #%0000011000000000,VDP_data

    move.l #$C01E0000,d0    ;Color 15 (Font)
    move.l d0,VDP_Ctrl
    move.w #%0000000011101110,VDP_data

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                   Set up Font
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; FONT IS 1BPP, THIS ROUTINE CONVERTS IT TO A 4BPP FORMAT.
    lea Font,A1                 ;Font Address in ROM
    move.l #Font_End-Font,d6                    ;Our font contains 96 letters 8 lines each

    move.l #$40000000,(VDP_Ctrl);Start writes to VRAM address $0000
NextFont:
    move.b (A1)+,d0     ;Get byte from font
    moveq.l #7,d5       ;Bit Count (8 bits)
    clr.l d1        ;Reset BuildUp Byte

Font_NextBit:           ;1 color per nibble = 4 bytes

    rol.l #3,d1     ;Shift BuildUp 3 bits left
    roxl.b #1,d0        ;Shift a Bit from the 1bpp font into the Pattern
    roxl.l #1,d1        ;Shift bit into BuildUp
    dbra D5,Font_NextBit    ;Next Bit from Font

    move.l d1,d0        ; Make fontfrom Color 1 to color 15
    rol.l #1,d1     ;Bit 1
    or.l d0,d1
    rol.l #1,d1     ;Bit 2
    or.l d0,d1
    rol.l #1,d1     ;Bit 3
    or.l d0,d1

    move.l d1,(VDP_Data);Write next Long of char (one line) to VDP
    dbra d6,NextFont    ;Loop until done



    clr.b Cursor_X      ;Clear Cursor XY
    clr.b Cursor_Y

    ;Turn on screen
    move.w  #$8144,(VDP_Ctrl);C00004 reg 1 = 0x44 unblank display

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; all of the above was just the prep work to boot the Sega Genesis, and had nothing to do with a 15 Puzzle.
; That's hardware for you!

    LEA GameRam,A0
        ;load the initial state of the puzzle. There is no randomization here unfortunately, as creating a
sufficient pseudo-RNG
        ;to make the game "believable" is more difficult than programming the game itself!
        ;so instead we'll start in such a manner that the player has to do quite a bit of work to win.
    MOVE.B #'F',(A0)+
    MOVE.B #'E',(A0)+
    MOVE.B #'D',(A0)+
    MOVE.B #'C',(A0)+
    MOVE.B #'B',(A0)+
    MOVE.B #'A',(A0)+
    MOVE.B #'9',(A0)+
    MOVE.B #'8',(A0)+
    MOVE.B #'7',(A0)+
    MOVE.B #'6',(A0)+
    MOVE.B #'5',(A0)+
    MOVE.B #'4',(A0)+
    MOVE.B #'3',(A0)+
    MOVE.B #'2',(A0)+
    MOVE.B #'1',(A0)+
    MOVE.B #' ',(A0)+
```

```
        ;puzzle will look like:
        ;FEDC
        ;BA98
        ;7654
        ;321

main:
    JSR Player_ReadControlsDual     ;get controller input
    move.w d0,(joypad1)


    ;adjust the number of these as you see fit.
    ;this affects the game's overall speed.
    JSR waitVBlank
    JSR waitVBlank
    JSR waitVBlank
    JSR waitVBlank
    JSR waitVBlank
    JSR waitVBlank
    JSR waitVBlank
    JSR waitVBlank
    JSR waitVBlank
    JSR waitVBlank

        ;find where the blank space is among GameRAM
    LEA GameRAM,a0
    MOVE.B #' ',D0
    JSR REPNE_SCASB
    MOVE.L A0,A1

;;;;;;;;;;;;;;;;;; check controller presses
JOYPAD_BITFLAG_M equ 2048
JOYPAD_BITFLAG_Z equ 1024
JOYPAD_BITFLAG_Y equ 512
JOYPAD_BITFLAG_X equ 256
JOYPAD_BITFLAG_S equ 128
JOYPAD_BITFLAG_C equ 64
JOYPAD_BITFLAG_B equ 32
JOYPAD_BITFLAG_A equ 16
JOYPAD_BITFLAG_R equ 8
JOYPAD_BITFLAG_L equ 4
JOYPAD_BITFLAG_D equ 2
JOYPAD_BITFLAG_U equ 1

JOYPAD_BITNUM_M equ 11
JOYPAD_BITNUM_Z equ 10
JOYPAD_BITNUM_Y equ 9
JOYPAD_BITNUM_X equ 8
JOYPAD_BITNUM_S equ 7
JOYPAD_BITNUM_C equ 6
JOYPAD_BITNUM_B equ 5
JOYPAD_BITNUM_A equ 4
JOYPAD_BITNUM_R equ 3
JOYPAD_BITNUM_L equ 2
JOYPAD_BITNUM_D equ 1
JOYPAD_BITNUM_U equ 0




    move.w (joypad1),D0

    BTST #JOYPAD_BITNUM_U,D0
    BNE JoyNotUp
        MOVEM.L D0/A1,-(SP)
            ADDA.L #4,A1
            CMPA.L #GameRam_End,A1
            BHI .doNothing
            ;OTHERWISE SWAP THE EMPTY SPACE WITH THE BYTE BELOW IT.
                MOVE.B (A1),D7
                MOVE.B (A0),(A1)
                MOVE.B D7,(A0)
.doNothing
        MOVEM.L (SP)+,D0/A1
        bra vdraw
JoyNotUp:
    BTST #JOYPAD_BITNUM_D,D0
    BNE JoyNotDown
```

```
        MOVEM.L D0/A1,-(SP)

            SUBA.L #4,A1        ;CHECK ONE ROW ABOVE WHERE WE ARE
            CMPA.L #GameRam,A1
            BCS .doNothing      ;if A1-4 IS BELOW THE START OF GAME RAM, DON'T MOVE
            ;OTHERWISE SWAP THE EMPTY SPACE WITH THE BYTE ABOVE IT.
                MOVE.B (A1),D7
                MOVE.B (A0),(A1)
                MOVE.B D7,(A0)
.doNothing:
        MOVEM.L (SP)+,D0/A1
        bra vdraw
JoyNotDown:
    BTST #JOYPAD_BITNUM_L,D0
    BNE JoyNotLeft
        MOVEM.L D0/A1,-(SP)
            ADDA.L #1,A1
            MOVE.L A1,D4
            MOVE.L A0,D3
            AND.L #3,D4
            AND.L #3,D3
            CMP.L D3,D4
            BCS .doNothing
            ;OTHERWISE SWAP THE EMPTY SPACE WITH THE BYTE TO THE LEFT
                MOVE.B (A1),D7
                MOVE.B (A0),(A1)
                MOVE.B D7,(A0)
.doNothing:
        MOVEM.L (SP)+,D0/A1
        bra vdraw
JoyNotLeft:
    BTST #JOYPAD_BITNUM_R,D0
    BNE JoyNotRight
        MOVEM.L D0/A1,-(SP)
            SUBA.L #1,A1
            MOVE.L A1,D4
            MOVE.L A0,D3
            AND.L #3,D4
            AND.L #3,D3
            CMP.L D3,D4
            BHI .doNothing
            ;OTHERWISE SWAP THE EMPTY SPACE WITH THE BYTE TO THE RIGHT
                MOVE.B (A1),D7
                MOVE.B (A0),(A1)
                MOVE.B D7,(A0)
.doNothing:
        MOVEM.L (SP)+,D0/A1
        bra vdraw
JoyNotRight:

vdraw:
    ;this actually draws the current state of the puzzle to the screen.
    LEA GameRam,A0
    CLR.B (Cursor_X)   ;reset text cursors to top left of screen
    CLR.B (Cursor_Y)

;draw the puzzle

;anything insize a REPT N...ENDR block is in-lined N times, back to back.
       rept 4

          MOVE.B (A0)+,D0
          JSR PrintChar

          MOVE.B (A0)+,D0
          JSR PrintChar

          MOVE.B (A0)+,D0
          JSR PrintChar

          MOVE.B (A0)+,D0
          JSR PrintChar         ;we just finished drawing one row of the puzzle. Now, begin a new line and
continue drawing.

              jsr newline
       endr
```

```
checkIfWin:
    ;YES THIS IS MESSY, I TRIED IT WITH A LOOP BUT IT WOULDN'T WORK SO I JUST UNROLLED THE LOOP.
    LEA GameRam,a4
    MOVE.B (A4)+,D5
    CMP.B #'1',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'2',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'3',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'4',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'5',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'6',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'7',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'8',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'9',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'A',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'B',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'C',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'D',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'E',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #'F',D5
    BNE .keepGoing

    MOVE.B (A4)+,D5
    CMP.B #' ',D5
    BNE .keepGoing

    clr.b (Cursor_X)
    move.b #7,(Cursor_Y)
    LEA victoryMessage,a3
    jsr PrintString
    jmp *  ;game freezes after you win.

.keepGoing:
;it's unlikely that the label "main" is in range of here so I didn't bother checking and just assumed it was out
of range.
;Otherwise I would have said "BEQ main" instead of BNE .keepGoing
    jmp main
```

```
VictoryMessage:
    DC.B "A WINNER IS YOU",255
    EVEN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
REPNE_SCASB:
    ;INPUT:
    ;A0 = POINTER TO START OF MEMORY
    ;D0 = THE BYTE TO SEARCH FOR
    ;OUTPUT = A0 POINTS TO THE BYTE THAT CONTAINED D0
    MOVE.B (A0),D1
    CMP.B D0,D1
    BEQ .done
    ADDA.L #1,A0
    BRA REPNE_SCASB
.done:
    RTS


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



Player_ReadControlsDual:

    move.b #%01000000,($A1000B) ; Set direction IOIIIIII (I=In O=Out)
    move.l #$A10003,a0      ;RW port for player 1

    move.b  #$40,(a0)   ; TH = 1
    nop     ;Delay
    nop
    move.b  (a0),d2     ; d0.b = --CBRLDU   Store in D2

    move.b  #$0,(a0)    ; TH = 0
    nop     ;Delay
    nop
    move.b  (a0),d1     ; d1.b = --SA--DU   Store in D1

    move.b  #$40,(a0)   ; TH = 1
    nop     ;Delay
    nop
    move.b  #$0,(a0)    ; TH = 0
    nop     ;Delay
    nop
    move.b  #$40,(a0)   ; TH = 1
    nop     ;Delay
    nop
    move.b  (a0),d3     ; d1.b = --CBXYZM   Store in D3
    move.b  #$0,(a0)    ; TH = 0

    clr.l d0            ;Clear buildup byte
    roxr.b d2
    roxr.b d0           ;U
    roxr.b d2
    roxr.b d0           ;D
    roxr.b d2
    roxr.b d0           ;L
    roxr.b d2
    roxr.b d0           ;R
    roxr.b #5,d1
    roxr.b d0           ;A
    roxr.b d2
    roxr.b d0           ;B
    roxr.b d2
    roxr.b d0           ;C
    roxr.b d1
    roxr.b d0           ;S

    move.l d3,d1
    roxl.l #7,d1        ;XYZ
    and.l #%0000011100000000,d1
    or.l d1,d0

    move.l d3,d1
```

```
    roxl.l #8,d1        ;M
    roxl.l #3,d1
    and.l #%0000100000000000,d1
    or.l d1,d0

    or.l #$FFFFF000,d0  ;Set unused bits to 1



    ;this returns player 1's buttons into D0 as the following:
    ;----MZYXSCBARLDU
    rts
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
waitVBlank:                         ;Bit 3 defines if we're in Vblank
    MOVE.L d0,-(sp)
.wait:
        move.w VDP_ctrl,d0
        and.w #%0000000000001000,d0     ;See if vblank is running
        bne .wait                   ;wait until it is

waitVBlank2:
        move.w VDP_ctrl,d0
        and.w #%0000000000001000,d0     ;See if vblank is running
        beq waitVBlank2                 ;wait until it isnt
    MOVE.L (SP)+,d0
    rts
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PrintChar:              ;Show D0 to screen
    moveM.l d0-d7/a0-a7,-(sp)
        and.l #$FF,d0           ;Keep only 1 byte
        sub #32,d0              ;No Characters in our font below 32
PrintCharAlt:
        Move.L  #$40000003,d5   ;top 4=write, bottom $3=Cxxx range
        clr.l d4                    ;Tilemap at $C000+

        Move.B (Cursor_Y),D4
        rol.L #8,D4             ;move $-FFF to $-FFF----
        rol.L #8,D4
        rol.L #7,D4             ;2 bytes per tile * 64 tiles per line
        add.L D4,D5             ;add $4------3

        Move.B (Cursor_X),D4
        rol.L #8,D4             ;move $-FFF to $-FFF----
        rol.L #8,D4
        rol.L #1,D4             ;2 bytes per tile
        add.L D4,D5             ;add $4------3

        MOVE.L  D5,(VDP_ctrl)   ; C00004 write next character to VDP
        MOVE.W  D0,(VDP_data)   ; C00000 store next word of name data

        addq.b #1,(Cursor_X)    ;INC Xpos
        move.b (Cursor_X),d0
        cmp.b #39,d0
        bls nextpixel_Xok
        jsr NewLine         ;If we're at end of line, start newline
nextpixel_Xok:
    moveM.l (sp)+,d0-d7/a0-a7
    rts

PrintString:
        move.b (a3)+,d0         ;Read a character in from A3
        cmp.b #255,d0
        beq PrintString_Done    ;return on 255
        jsr PrintChar           ;Print the Character
        bra PrintString
PrintString_Done:
    rts

NewLine:
    addq.b #1,(Cursor_Y)        ;INC Y
    clr.b (Cursor_X)            ;Zero X
    rts

Font:
;1bpp font - 8x8 96 characters
;looks just like your typical "8-bit" font. You'll just have to take my word for it.
     DC.B $00,$00,$00,$00,$00,$00,$00,$00,$18,$3c,$3c,$18,$18,$00,$18,$18
```

```
      DC.B $36,$36,$12,$24,$00,$00,$00,$00,$00,$12,$7f,$24,$24,$fe,$48,$00
      DC.B $00,$04,$1e,$28,$1c,$0a,$3c,$10,$00,$62,$64,$08,$10,$26,$46,$00
      DC.B $00,$18,$24,$20,$12,$2c,$44,$3a,$18,$18,$08,$10,$00,$00,$00,$00
      DC.B $08,$10,$20,$20,$20,$20,$10,$08,$10,$08,$04,$04,$04,$04,$08,$10
      DC.B $00,$10,$38,$10,$28,$00,$00,$00,$00,$00,$10,$10,$7c,$10,$10,$00
      DC.B $00,$00,$00,$00,$0c,$0c,$04,$08,$00,$00,$00,$00,$7e,$00,$00,$00
      DC.B $00,$00,$00,$00,$00,$18,$18,$00,$01,$02,$04,$08,$10,$20,$40,$00
      DC.B $1c,$26,$63,$63,$63,$32,$1c,$00,$0c,$1c,$0c,$0c,$0c,$0c,$3f,$00
      DC.B $3e,$63,$07,$1e,$3c,$70,$7f,$00,$3f,$06,$0c,$1e,$03,$63,$3e,$00
      DC.B $0e,$1e,$36,$66,$7f,$06,$06,$00,$7e,$60,$7e,$03,$03,$63,$3e,$00
      DC.B $1e,$30,$60,$7e,$63,$63,$3e,$00,$7f,$63,$06,$0c,$18,$18,$18,$00
      DC.B $3c,$62,$72,$3c,$4f,$43,$3e,$00,$3e,$63,$63,$3f,$03,$06,$3c,$00
      DC.B $00,$18,$18,$00,$18,$18,$00,$00,$00,$0c,$0c,$00,$0c,$0c,$04,$08
      DC.B $00,$00,$06,$18,$60,$18,$06,$00,$00,$00,$00,$7e,$00,$7e,$00,$00
      DC.B $00,$00,$60,$18,$06,$18,$60,$00,$1c,$36,$36,$06,$0c,$00,$0c,$0c
      DC.B $3c,$42,$99,$a1,$a1,$99,$42,$3c,$1c,$36,$63,$63,$7f,$63,$63,$00
      DC.B $7e,$63,$63,$7e,$63,$63,$7e,$00,$1e,$33,$60,$60,$60,$33,$1e,$00
      DC.B $7c,$66,$63,$63,$63,$66,$7c,$00,$3f,$30,$30,$3e,$30,$30,$3f,$00
      DC.B $7f,$60,$60,$7e,$60,$60,$60,$00,$1f,$30,$60,$67,$63,$33,$1f,$00
      DC.B $63,$63,$63,$7f,$63,$63,$63,$00,$3f,$0c,$0c,$0c,$0c,$0c,$3f,$00
      DC.B $03,$03,$03,$03,$03,$63,$3e,$00,$63,$66,$6c,$78,$7c,$6e,$67,$00
      DC.B $30,$30,$30,$30,$30,$30,$3f,$00,$63,$77,$7f,$7f,$6b,$63,$63,$00
      DC.B $63,$73,$7b,$7f,$6f,$67,$63,$00,$3e,$63,$63,$63,$63,$63,$3e,$00
      DC.B $7e,$63,$63,$63,$7e,$60,$60,$00,$3e,$63,$63,$63,$6f,$66,$3d,$00
      DC.B $7e,$63,$63,$67,$7c,$6e,$67,$00,$3c,$66,$60,$3e,$03,$63,$3e,$00
      DC.B $3f,$0c,$0c,$0c,$0c,$0c,$0c,$00,$63,$63,$63,$63,$63,$63,$3e,$00
      DC.B $63,$63,$63,$77,$3e,$1c,$08,$00,$63,$63,$6b,$7f,$7f,$77,$63,$00
      DC.B $63,$77,$3e,$1c,$3e,$77,$63,$00,$33,$33,$33,$1e,$0c,$0c,$0c,$00
      DC.B $7f,$07,$0e,$1c,$38,$70,$7f,$00,$00,$38,$20,$20,$20,$38,$00
      DC.B $80,$40,$20,$10,$08,$04,$02,$00,$00,$1c,$04,$04,$04,$04,$1c,$00
      DC.B $10,$28,$44,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$7e,$00
      DC.B $00,$20,$10,$00,$00,$00,$00,$00,$00,$18,$04,$1c,$24,$2c,$1c,$00
      DC.B $00,$20,$20,$38,$24,$24,$38,$00,$00,$00,$1c,$20,$20,$20,$1c,$00
      DC.B $00,$04,$04,$1c,$24,$24,$1c,$00,$00,$00,$1c,$24,$3c,$20,$1c,$00
      DC.B $00,$18,$24,$20,$30,$20,$20,$00,$00,$1c,$24,$24,$1c,$04,$3c,$00
      DC.B $00,$20,$20,$38,$24,$24,$24,$00,$00,$10,$00,$10,$10,$10,$10,$00
      DC.B $08,$00,$08,$08,$08,$08,$28,$10,$20,$20,$24,$28,$30,$28,$24,$00
      DC.B $10,$10,$10,$10,$10,$10,$18,$00,$00,$00,$40,$68,$54,$54,$54,$00
      DC.B $00,$00,$28,$34,$24,$24,$24,$00,$00,$00,$1c,$22,$22,$22,$1c,$00
      DC.B $00,$00,$38,$24,$24,$38,$20,$20,$00,$00,$1c,$24,$24,$1c,$04,$04
      DC.B $00,$00,$2c,$30,$20,$20,$20,$00,$00,$00,$1c,$20,$1c,$02,$3c,$00
      DC.B $00,$10,$3c,$10,$10,$14,$08,$00,$00,$00,$24,$24,$24,$24,$1a,$00
      DC.B $00,$00,$24,$24,$24,$14,$18,$00,$00,$00,$92,$92,$92,$5a,$6c,$00
      DC.B $00,$00,$22,$14,$08,$14,$22,$00,$00,$00,$24,$24,$1c,$04,$18,$00
      DC.B $00,$00,$3c,$04,$18,$20,$3c,$00,$00,$08,$10,$10,$20,$10,$10,$08
      DC.B $18,$18,$18,$18,$18,$18,$18,$18,$00,$10,$08,$08,$04,$08,$08,$10
      DC.B $00,$00,$00,$30,$4a,$04,$00,$00,$1c,$7f,$00,$7f,$55,$55,$55,$00
Font_End:

VDPSettings:
    DC.B $04 ; 0 mode register 1                                          ---H-1M-
    DC.B $04 ; 1 mode register 2                                          -DVdP---
    DC.B $30 ; 2 name table base for scroll A (A=top 3 bits)              --AAA--- = $C000
    DC.B $3C ; 3 name table base for window (A=top 4 bits / 5 in H40 Mode)  --AAAAA- = $F000
    DC.B $07 ; 4 name table base for scroll B (A=top 3 bits)              -----AAA = $E000
    DC.B $6C ; 5 sprite attribute table base (A=top 7 bits / 6 in H40)    -AAAAAAA = $D800
    DC.B $00 ; 6 unused register                                          --------
    DC.B $00 ; 7 background color (P=Palette C=Color)                     --PPCCCC
    DC.B $00 ; 8 unused register                                          --------
    DC.B $00 ; 9 unused register                                          --------
    DC.B $FF ;10 H interrupt register (L=Number of lines)                 LLLLLLLL
    DC.B $00 ;11 mode register 3                                          ----IVHL
    DC.B $81 ;12 mode register 4 (C bits both1 = H40 Cell)                C---SIIC
    DC.B $37 ;13 H scroll table base (A=Top 6 bits)                       --AAAAAA = $FC00
    DC.B $00 ;14 unused register                                          --------
    DC.B $02 ;15 auto increment (After each Read/Write)                   NNNNNNNN
    DC.B $01 ;16 scroll size (Horiz & Vert size of ScrollA & B)           --VV--HH = 64x32 tiles
    DC.B $00 ;17 window H position (D=Direction C=Cells)                  D--CCCCC
    DC.B $00 ;18 window V position (D=Direction C=Cells)                  D--CCCCC
    DC.B $FF ;19 DMA length count low                                     LLLLLLLL
    DC.B $FF ;20 DMA length count high                                    HHHHHHHH
    DC.B $00 ;21 DMA source address low                                   LLLLLLLL
    DC.B $00 ;22 DMA source address mid                                   MMMMMMMM
    DC.B $80 ;23 DMA source address high (C=CMD)                          CCHHHHHH
VDPSettingsEnd:
    even
```

Output:

Screenshot of emulator (https://ibb.co/4MZpL4W)

AArch64 Assembly

Works with: as version Raspberry Pi 3B version Buster 64 bits

```
/* ARM assembly AARCH64 Raspberry PI 3B */
/*  program puzzle15_64.s   */

/*******************************************/
/* Constantes file                         */
/*******************************************/
/* for this file see task include a file in language AArch64 assembly*/
.include "../includeConstantesARM64.inc"

.equ NBBOX,  16
.equ GRAINE,  123456      // change for other game
.equ NBSHUFFLE, 4
.equ KEYSIZE,  8

.equ IOCTL,     0x1D  // Linux syscall
.equ SIGACTION, 0x86   // Linux syscall
.equ SYSPOLL,   0x16  // Linux syscall
.equ CREATPOLL, 0x14  // Linux syscall
.equ CTLPOLL,   0x15  // Linux syscall

.equ TCGETS,    0x5401
.equ TCSETS,    0x5402
.equ ICANON,    2
.equ ECHO,     10
.equ POLLIN,    1
.equ EPOLL_CTL_ADD,    1

.equ SIGINT,   2    // Issued if the user sends an interrupt signal (Ctrl + C)
.equ SIGQUIT,  3    // Issued if the user sends a quit signal (Ctrl + D)
.equ SIGTERM, 15    // Software termination signal (sent by kill by default)
.equ SIGTTOU, 22    //

/*******************************************/
/* Structures                              */
/*******************************************/
/* structure termios see doc linux*/
    .struct  0
term_c_iflag:                   // input modes
    .struct  term_c_iflag + 4
term_c_oflag:                   // output modes
    .struct  term_c_oflag + 4
term_c_cflag:                   // control modes
    .struct  term_c_cflag + 4
term_c_lflag:                   // local modes
    .struct  term_c_lflag + 4
term_c_cc:                      // special characters
    .struct  term_c_cc + 20     // see length if necessary
term_fin:

/* structure sigaction see doc linux */
    .struct  0
sa_handler:
    .struct  sa_handler + 8
sa_mask:
    .struct  sa_mask + 8
sa_flags:
    .struct  sa_flags + 8
sa_sigaction:
    .struct  sa_sigaction + 8
sa_fin:

/* structure poll see doc linux */
    .struct  0
poll_event:
    .struct  poll_event + 8
```

```
poll_fd:                            //   File Descriptor
    .struct  poll_fd + 8
poll_fin:
/*********************************/
/* Initialized data              */
/*********************************/
.data
sMessResult:           .ascii " "
sMessValeur:           .fill 11, 1, ' '             // size => 11
szCarriageReturn:      .asciz "\n"
szMessGameWin:         .asciz "You win in @ move number !!!!\n"
szMessMoveError:       .asciz "Huh... Impossible move !!!!\n"
szMessErreur:          .asciz "Error detected.\n"
szMessErrInitTerm:     .asciz "Error terminal init.\n"
szMessErrInitPoll:     .asciz "Error poll init.\n"
szMessErreurKey:       .asciz "Error read key.\n"
szMessSpaces:          .asciz "    "
qGraine:               .quad GRAINE
szMessErr:             .asciz    "Error code hexa : @ décimal : @ \n"

szClear:     .byte 0x1B
             .byte 'c'                          // console clear
             .byte 0
/*********************************/
/* UnInitialized data            */
/*********************************/
.bss
.align 4
sZoneConv:      .skip 24
qCodeError:     .skip 8
ibox:           .skip 4 * NBBOX                // game boxes
qEnd:           .skip 8                        // 0 loop  1 = end loop
qTouche:        .skip KEYSIZE                  // value key pressed
stOldtio:       .skip term_fin                 // old terminal state
stCurtio:       .skip term_fin                 // current terminal state
stSigAction:    .skip sa_fin                   // area signal structure
stSigAction1:   .skip sa_fin
stSigAction2:   .skip sa_fin
stSigAction3:   .skip sa_fin
stPoll1:        .skip poll_fin                 // area poll structure
stPoll2:        .skip poll_fin
stevents:       .skip 16
/*********************************/
/*  code section                 */
/*********************************/
.text
.global main
main:                               // entry of program
    mov x0,#0
    bl initTerm                     // terminal init
    cmp x0,0                        // error ?
    blt 100f
    bl initPoll                     // epoll instance init
    cmp x0,0
    blt 99f
    mov x22,x0                      // save epfd
    ldr x2,qAdribox
    mov x9,#0                       // init counter
    mov x0,0
1:                                  // loop init boxs
    add x1,x0,#1                    // box value
    str w1,[x2,x0, lsl #2]          // store value
    add x0,x0,#1                    // increment counter
    cmp x0,#NBBOX - 2               // end ?
    ble 1b
    mov x10,#15                     // empty box location
    ldr x0,qAdribox
    bl shuffleGame
2:                                  // loop moves
    ldr x0,qAdribox
    bl displayGame
3:
    mov x0,x22                      // epfd
    bl waitKey
    cmp x0,0
    beq 3b                          // no ket pressed -> loop
    blt 99f                         // error ?
    bl readKey                      // read key
```

```
    cmp x0,#-1
    beq 99f                         // error
    cmp x0,3                        // <ctrl_C>
    beq 5f
    cmp x0,113                      // saisie q (quit) ?
    beq 5f
    cmp x0,81                       // saisie Q  (Quit)?
    beq 5f
    mov x1,x0                       // key
    ldr x0,qAdribox
    bl keyMove                      // analyze key move
    ldr x0,qAdribox
    bl gameOK                       // end game ?
    cmp x0,#1
    bne 2b                          // no -> loop
                                    // win
    mov x0,x9                       // move counter
    ldr x1,qAdrsZoneConv
    bl conversion10
    ldr x0,qAdrszMessGameWin
    ldr x1,qAdrsZoneConv
    bl strInsertAtCharInc           // insert result at @ character
    bl affichageMess
5:
    bl restauTerm                   // terminal restaur
    mov x0, #0                      // return code
    b 100f
99:
    bl restauTerm                   // terminal restaur
    mov x0,1                        // return code error
    b 100f
100:                                // standard end of the program
    mov x8, #EXIT                   // request to exit program
    svc #0                          // perform the system call

qAdrsMessValeur:          .quad sMessValeur
qAdrszCarriageReturn:     .quad szCarriageReturn
qAdrsMessResult:          .quad sMessResult
qAdribox:                 .quad ibox
qAdrszMessGameWin:        .quad szMessGameWin
qAdrstevents:             .quad stevents
qAdrszMessErreur:         .quad szMessErreur
qAdrstOldtio:             .quad stOldtio
qAdrstCurtio:             .quad stCurtio
qAdrstSigAction:          .quad stSigAction
qAdrstSigAction1:         .quad stSigAction1
qAdrSIG_IGN:              .quad 1
qAdrqEnd:                 .quad qEnd
qAdrqTouche:              .quad qTouche
qAdrszMessErrInitTerm:    .quad szMessErrInitTerm
qAdrszMessErrInitPoll:    .quad szMessErrInitPoll
qAdrszMessErreurKey:      .quad szMessErreurKey
/******************************************************************/
/*     key move                                                   */
/******************************************************************/
/* x0 contains boxs address           */
/* x1 contains key value               */
/* x9 move counter                     */
/* x10 contains location empty box    */
keyMove:
    stp x1,lr,[sp,-16]!             // save  registers
    mov x7,x0
    lsr x1,x1,16
    cmp x1,#0x42                    // down arrow
    bne 1f
    cmp x10,#4                      // if x10 < 4   error
    blt 80f
    sub x2,x10,#4                   // compute location
    b 90f
1:
    cmp x1,#0x41                    // high arrow
    bne 2f
    cmp x10,#11                     // if x10 > 11   error
    bgt 80f
    add x2,x10,#4                   // compute location
    b 90f
2:
    cmp x1,#0x43                    // right arrow
```

```
    bne 3f
    tst x10,#0b11                   // if x10 = 0,4,8,12   error
    beq 80f
    sub x2,x10,#1                   // compute location
    b 90f
3:
    cmp x1,#0x44                    // left arrow
    bne 100f
    and x3,x10,#0b11                // error if x10 = 3 7 11 and 15
    cmp x3,#3
    beq 80f
    add x2,x10,#1                   // compute location
    b 90f

80:                                 // move error
    ldr x0,qAdrqCodeError
    mov x1,#1
    str x1,[x0]
    b 100f
90:                                 // white box and move box inversion
    ldr w3,[x7,x2,lsl #2]
    str w3,[x7,x10,lsl #2]
    mov x10,x2
    mov x3,#0
    str w3,[x7,x10,lsl #2]
    add x9,x9,#1                       // increment move counter
100:
    ldp x1,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30
qAdrqCodeError:             .quad qCodeError
/******************************************************************/
/*     shuffle game                                               */
/******************************************************************/
/* x0 contains boxs address           */
shuffleGame:
    stp x1,lr,[sp,-16]!            // save  registers
    stp x2,x3,[sp,-16]!            // save  registers
    stp x4,x5,[sp,-16]!            // save  registers
    mov x1,x0
    mov x0,NBSHUFFLE
    bl genereraleas
    lsl x4,x0,#1
1:
    mov x0,#14
    bl genereraleas
    add x3,x0,#1
    mov x0,#14
    bl genereraleas
    add x5,x0,#1
    ldr w2,[x1,x3,lsl #2]
    ldr w0,[x1,x5,lsl #2]
    str w2,[x1,x5,lsl #2]
    str w0,[x1,x3,lsl #2]
    subs x4,x4,#1
    bgt 1b

100:
    ldp x4,x5,[sp],16              // restaur  2 registers
    ldp x2,x3,[sp],16              // restaur  2 registers
    ldp x1,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30
/******************************************************************/
/*     game Ok ?                                                  */
/******************************************************************/
/* x0 contains boxs address           */
gameOK:
    stp x1,lr,[sp,-16]!            // save  registers
    stp x2,x3,[sp,-16]!            // save  registers
    mov x2,#0
    ldr w3,[x0,x2,lsl #2]
    add x2,x2,#1
1:
    ldr w1,[x0,x2,lsl #2]
    cmp w1,w3
    bge 2f
    mov x0,#0                      // game not Ok
    b 100f
2:
```

```
    mov x3,x1
    add x2,x2,#1
    cmp x2,#NBBOX -2
    ble 1b
    mov x0,#                    // game Ok

100:
    ldp x2,x3,[sp],16           // restaur  2 registers
    ldp x1,lr,[sp],16           // restaur  2 registers
    ret                         // return to address lr x30
/******************************************************************/
/*     display game                                               */
/******************************************************************/
/* x0 contains boxs address           */
displayGame:
    stp x1,lr,[sp,-16]!         // save  registers
                                // clear screen !
    mov x4,x0
    ldr x0,qAdrszClear
    bl affichageMess
    mov x2,#0
    ldr x1,qAdrsMessValeur
1:
    ldr w0,[x4,x2,lsl #2]
    cmp w0,#0
    bne 2f
    ldr w0,iSpaces                 // store spaces
    str w0,[x1]
    b 3f
2:
    bl conversion10                  // call conversion decimal
    cmp x0,1
    beq 21f
    mov x0,0x20
    strh w0,[x1,#2]
    b 3f
21:
    mov w0,0x2020
    str w0,[x1,#1]
3:
    ldr x0,qAdrsMessResult
    bl affichageMess                 // display message
    add x0,x2,#1
    tst x0,#0b11
    bne 4f
    ldr x0,qAdrszCarriageReturn
    bl affichageMess                 // display message
4:
    add x2,x2,#1
    cmp x2,#NBBOX - 1
    ble 1b
    ldr x0,qAdrszCarriageReturn
    bl affichageMess                 // display line return
    ldr x0,qAdrqCodeError            // error detected ?
    ldr x1,[x0]
    cmp x1,#0
    beq 100f
    mov x1,#0                        // raz error code
    str x1,[x0]
    ldr x0,qAdrszMessMoveError       // display error message
    bl affichageMess
100:
    ldp x1,lr,[sp],16                // restaur  2 registers
    ret                              // return to address lr x30
iSpaces:              .int 0x00202020   // spaces
qAdrszClear:          .quad szClear
qAdrszMessMoveError:  .quad szMessMoveError

/************************************************/
/*   Generation random number                   */
/************************************************/
/* x0 contains limit  */
genereraleas:
    stp x1,lr,[sp,-16]!         // save  registers
    stp x2,x3,[sp,-16]!         // save  registers
    ldr x1,qAdrqGraine
    ldr x2,[x1]
    ldr x3,qNbDep1
```

```
    mul x2,x3,x2
    ldr x3,qNbDep2
    add x2,x2,x3
    str x2,[x1]                    // maj de la graine pour l appel suivant
    cmp x0,#0
    beq 100f
    udiv x3,x2,x0
    msub x0,x3,x0,x2               // résult = remainder

100:                               // end function
    ldp x2,x3,[sp],16              // restaur  2 registers
    ldp x1,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30
/***************************************************/
qAdrqGraine: .quad qGraine
qNbDep1:     .quad 0x0019660d
qNbDep2:     .quad 0x3c6ef35f

/******************************************************************/
/*     traitement du signal                                       */
/******************************************************************/
sighandler:
    stp x0,lr,[sp,-16]!            // save  registers
    str x1,[sp,-16]!
    ldr x0,qAdrqEnd
    mov x1,#1                      // maj zone end
    str x1,[x0]
    ldr x1,[sp],16
    ldp x0,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30
/***************************************************/
/*   display error message                         */
/***************************************************/
/* x0 contains error code  x1 : message address */
displayError:
    stp x2,lr,[sp,-16]!            // save  registers
    mov x2,x0                      // save error code
    mov x0,x1
    bl affichageMess
    mov x0,x2                      // error code
    ldr x1,qAdrsZoneConv
    bl conversion16                // conversion hexa
    ldr x0,qAdrszMessErr           // display error message
    ldr x1,qAdrsZoneConv
    bl strInsertAtCharInc              // insert result at @ character
    mov x3,x0
    mov x0,x2                      // error code
    ldr x1,qAdrsZoneConv               // result address
    bl conversion10S                // conversion decimale
    mov x0,x3
    ldr x1,qAdrsZoneConv
    bl strInsertAtCharInc              // insert result at @ character
    bl affichageMess
100:
    ldp x2,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30
qAdrszMessErr:                 .quad szMessErr
qAdrsZoneConv:                 .quad sZoneConv
/*********************************/
/* init terminal state            */
/*********************************/
initTerm:
    stp x1,lr,[sp,-16]!            // save  registers
    /* read terminal state */
    mov x0,STDIN                   // input console
    mov x1,TCGETS
    ldr x2,qAdrstOldtio
    mov x8,IOCTL                   // call system Linux
    svc 0
    cbnz x0,98f                    // error ?

    adr x0,sighandler              // adresse routine traitement signal
    ldr x1,qAdrstSigAction         // adresse structure sigaction
    str x0,[x1,sa_handler]         // maj handler
    mov x0,SIGINT                  // signal type
    ldr x1,qAdrstSigAction
    mov x2,0
    mov x3,8
```

```
    mov x8,SIGACTION                // call system
    svc 0

    cmp x0,0                        // error ?
    bne 98f
    mov x0,SIGQUIT
    ldr x1,qAdrstSigAction
    mov x2,0                        // NULL
    mov x8,SIGACTION                // call system
    svc 0
    cmp x0,0                        // error ?
    bne 98f
    mov x0,SIGTERM
    ldr x1,qAdrstSigAction
    mov x2,0                        // NULL
    mov x8,SIGACTION                // appel systeme
    svc 0
    cmp x0,0
    bne 98f
    //
    adr x0,qAdrSIG_IGN              // address signal igonre function
    ldr x1,qAdrstSigAction1
    str x0,[x1,sa_handler]
    mov x0,SIGTTOU                  //invalidate other process signal
    ldr x1,qAdrstSigAction1
    mov x2,0                        // NULL
    mov x8,SIGACTION                // call system
    svc 0
    cmp x0,0
    bne 98f
    //
    /* read terminal current state  */
    mov x0,STDIN
    mov x1,TCGETS
    ldr x2,qAdrstCurtio             // address current termio
    mov x8,IOCTL                    // call systeme
    svc 0
    cmp x0,0                        // error ?
    bne 98f
    mov x2,ICANON | ECHO            // no key pressed echo on display
    mvn x2,x2                       // and one key
    ldr x1,qAdrstCurtio
    ldr x3,[x1,#term_c_lflag]
    and x3,x2,x2                    // add flags
    str x3,[x1,#term_c_lflag]       // and store
    mov x0,STDIN                    // maj terminal current state
    mov x1,TCSETS
    ldr x2,qAdrstCurtio
    mov x8,IOCTL                    // call system
    svc 0
    cbz x0,100f
98:                                 // error display
    ldr x1,qAdrszMessErrInitTerm
    bl   displayError
    mov x0,-1
100:
    ldp x1,lr,[sp],16               // restaur  2 registers
    ret                             // return to address lr x30
qAdrstSigAction2:    .quad stSigAction2
qAdrstSigAction3:    .quad stSigAction3
/*********************************/
/* init instance epool            */
/*********************************/
initPoll:
    stp x1,lr,[sp,-16]!             // save  registers
    ldr x0,qAdrstevents
    mov x1,STDIN                    // maj structure events
    str x1,[x0,#poll_fd]            // maj FD
    mov x1,POLLIN                   // action code
    str x1,[x0,#poll_event]
    mov x0,0
    mov x8,CREATPOLL                // create epoll instance
    svc 0
    cmp x0,0                        // error ?
    ble 98f
    mov x10,x0                      // return FD epoll instance
    mov x1,EPOLL_CTL_ADD
    mov x2,STDIN                    // Fd to we want add
```

```
    ldr x3,qAdrstevents            // structure events address
    mov x8,CTLPOLL                 // call system control epoll
    svc 0
    cmp x0,0                       // error ?
    blt 98f                       // no
    mov x0,x10                     // return FD epoll instance
    b 100f
98:                                // error display
    ldr x1,qAdrszMessErrInitPoll   // error message
    bl   displayError
    mov x0,-1
100:
    ldp x1,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30
/*********************************/
/* wait key                      */
/*********************************/
/* x0 contains FD poll    */
waitKey:
    stp x1,lr,[sp,-16]!            // save  registers
    ldr x11,qAdrqTouche            // key address
    str xzr,[x11]                  // raz key
1:
    ldr x1,qAdrqEnd                // if signal ctrl-c  -> end
    ldr x1,[x1]
    cbnz x1,100f

    ldr x1,qAdrstevents
    mov x2,12                      // size events
    mov x3,1                       // timeout = 1  TODO: ??
    mov x4,0
    mov x8,SYSPOLL                 // call system wait POLL
    svc 0
    cmp x0,0                       // key pressed ?
    bge 100f
98:                                // error display
    ldr x1,qAdrszMessErreurKey        // error message
    bl   displayError
    mov x0,-1
100:
    ldp x1,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30
/*********************************/
/* read key                      */
/*********************************/
/* x0 returns key value */
readKey:
    stp x1,lr,[sp,-16]!            // save  registers
    mov x0,STDIN                   // File Descriptor
    ldr x1,qAdrqTouche             // buffer address
    mov x2,KEYSIZE                 // key size
    mov x8,READ                    // read key
    svc #0
    cmp x0,0                       // error ?
    ble 98f
    ldr x2,qAdrqTouche             // key address
    ldr x0,[x2]
    b 100f
98:                                // error display
    ldr x1,qAdrszMessErreur        // error message
    bl   displayError
    mov x0,-1
100:
    ldp x1,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30
/*********************************/
/* restaur terminal state        */
/*********************************/
restauTerm:
    stp x1,lr,[sp,-16]!            // save  registers
    mov x0,STDIN                   // end then restaur begin state terminal
    mov x1,TCSETS
    ldr x2,qAdrstOldtio
    mov x8,IOCTL                   // call system
    svc 0
    cbz x0,100f
    ldr x1,qAdrszMessErreur        // error message
    bl   displayError
```

```
100:
    ldp x1,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30

/******************************************************/
/*        File Include fonctions                      */
/******************************************************/
/* for this file see task include a file in language AArch64 assembly */
.include "../includeARM64.inc"
```

Action!

```
DEFINE BOARDSIZE="16"
DEFINE X0="13"
DEFINE Y0="6"
DEFINE ITEMW="3"
DEFINE ITEMH="2"

BYTE ARRAY board(BOARDSIZE)
BYTE emptyX,emptyY,solved,first=[1]

BYTE FUNC Index(BYTE x,y)
RETURN (x+y*4)

PROC UpdateItem(BYTE x,y)
  BYTE item

  Position(X0+x*ITEMW+1,Y0+y*ITEMH+1)
  item=board(Index(x,y))
  IF item=0 THEN
    Print("  ")
  ELSEIF item<10 THEN
    Put(160) Put(item+176)
  ELSE
    Put(item/10+176)
    Put(item MOD 10+176)
  FI
RETURN

PROC UpdateBoard()
  BYTE x,y

  FOR y=0 TO 3
  DO
    FOR x=0 TO 3
    DO
      UpdateItem(x,y)
    OD
  OD
RETURN

PROC DrawGrid()
  CHAR ARRAY
    top=[13 17 18 18 23 18 18 23 18 18 23 18 18 5],
    row=[13 124 32 32 124 32 32 124 32 32 124 32 32 124],
    mid=[13 1 18 18 19 18 18 19 18 18 19 18 18 4],
    bot=[13 26 18 18 24 18 18 24 18 18 24 18 18 3]
  BYTE y,i

  y=Y0
  Position(X0,y) Print(top) y==+1
  Position(X0,y) Print(row) y==+1
  FOR i=0 TO 2
  DO
    Position(X0,y) Print(mid) y==+1
    Position(X0,y) Print(row) y==+1
  OD
  Position(X0,y) Print(bot)
RETURN

PROC DrawBoard()
  DrawGrid()
  UpdateBoard()
RETURN
```

```
PROC FindEmpty()
  BYTE i

  FOR i=0 TO BOARDSIZE-1
  DO
    IF board(i)=0 THEN
      emptyX=i MOD 4
      emptyY=i/4
    FI
  OD
RETURN

PROC Wait(BYTE frames)
  BYTE RTCLOK=$14
  frames==+RTCLOK
  WHILE frames#RTCLOK DO OD
RETURN

PROC UpdateStatus()
  Position(9,3) Print("Game status: ")
  IF solved THEN
    Print("SOLVED !")
    IF first=0 THEN
      Sound(0,100,10,5) Wait(5)
      Sound(0,60,10,5) Wait(5)
      Sound(0,40,10,5) Wait(5)
      Sound(0,0,0,0)
    FI
    first=0
  ELSE
    Print("Shuffled")
  FI
RETURN

PROC InitBoard()
  BYTE i

  FOR i=1 TO BOARDSIZE
  DO
    board(i-1)=i MOD 16
  OD
  FindEmpty()
  solved=1
  UpdateStatus()
RETURN

BYTE FUNC IsSolved()
  BYTE i

  FOR i=1 TO BOARDSIZE
  DO
    IF board(i-1)#i MOD 16 THEN
      RETURN (0)
    FI
  OD
RETURN (1)

PROC CheckStatus()
  BYTE tmp

  tmp=IsSolved()
  IF solved#tmp THEN
    solved=tmp
    UpdateStatus()
  FI
RETURN

PROC Swap(BYTE x1,y1,x2,y2)
  BYTE tmp,i1,i2

  i1=Index(x1,y1)
  i2=Index(x2,y2)
  tmp=board(i1)
  board(i1)=board(i2)
  board(i2)=tmp
  UpdateItem(x1,y1)
  UpdateItem(x2,y2)
```

```
  CheckStatus()
RETURN

PROC Shuffle()
  BYTE i,j,tmp

  i=BOARDSIZE-1
  WHILE i>0
  DO
    j=Rand(i)
    tmp=board(i)
    board(i)=board(j)
    board(j)=tmp
    i==-1
  OD
  FindEmpty()
  UpdateBoard()
  CheckStatus()
RETURN

PROC MoveLeft()
  IF emptyX=0 THEN RETURN FI
  Swap(emptyX,emptyY,emptyX-1,emptyY)
  emptyX==-1
RETURN

PROC MoveRight()
  IF emptyX=3 THEN RETURN FI
  Swap(emptyX,emptyY,emptyX+1,emptyY)
  emptyX==+1
RETURN

PROC MoveUp()
  IF emptyY=0 THEN RETURN FI
  Swap(emptyX,emptyY,emptyX,emptyY-1)
  emptyY==-1
RETURN

PROC MoveDown()
  IF emptyY=3 THEN RETURN FI
  Swap(emptyX,emptyY,emptyX,emptyY+1)
  emptyY==+1
RETURN

PROC Main()
  BYTE k,lastStick=[255],currStick,
    CH=$02FC, ;Internal hardware value for last key pressed
    CRSINH=$02F0 ;Controls visibility of cursor

  Graphics(0)
  SetColor(2,0,2)
  CRSINH=1 ;hide cursor
  Position(10,18) Print("Joystick - move tiles")
  Position(9,19) Print("Space bar - shuffle")
  Position(15,20) Print("ESC - exit")
  InitBoard()
  DrawBoard()
  DO
    currStick=Stick(0)
    IF currStick#lastStick THEN
      IF currStick=11 THEN MoveRight()
      ELSEIF currStick=7 THEN MoveLeft()
      ELSEIF currStick=13 THEN MoveUp()
      ELSEIF currStick=14 THEN MoveDown()
      FI
    FI
    lastStick=currStick
    k=CH
    IF k#$FF THEN CH=$FF FI
    IF k=33 THEN Shuffle()
    ELSEIF k=28 THEN EXIT
    FI
  OD
RETURN
```

Output:

Screenshot from Atari 8-bit computer (https://gitlab.com/amarok8bit/action-rosetta-code/-/raw/master/images/15_puzzle_game.png)

Ada

We fist define a generic package Generic_Puzzle. Upon instantiation, it can take any number of rows, any number of columns for a rows*columns-1 game. Instead of plain numbers, the tiles on the board can have arbitrary names (but they should all be of the same length). The package user can request the name for the tile at a certain (row,column)-point, and the set of possible moves. The user can move the empty space up, down, left and right (if possible). If the user makes the attempt to perform an impossible move, a Constraint_Error is raised.

```
generic
   Rows, Cols: Positive;
   with function Name(N: Natural) return String; -- with Pre => (N < Rows*Cols);
   -- Name(0) shall return the name for the empty tile
package Generic_Puzzle is

   subtype Row_Type is Positive range 1 .. Rows;
   subtype Col_Type is Positive range 1 .. Cols;
   type Moves is (Up, Down, Left, Right);
   type Move_Arr is array(Moves) of Boolean;

   function Get_Point(Row: Row_Type; Col: Col_Type) return String;
   function Possible return Move_Arr;
   procedure Move(The_Move: Moves);

end Generic_Puzzle;
```

The package implementation is as follows.

```
package body Generic_Puzzle is

   Field: array(Row_Type, Col_Type) of Natural;
   Current_R: Row_Type := Rows;
   Current_C: Col_Type := Cols;
   -- invariant: Field(Current_R, Current_C=0)
   -- and for all R, C: Field(R, C) < R*C
   -- and for all (R, C) /= (RR, CC): Field(R, C) /= Field(RR, CC)

   function Get_Point(Row: Row_Type; Col: Col_Type) return String is
      (Name(Field(Row, Col)));

   function Possible return Move_Arr is
      (Up => Current_R > 1, Down => Current_R < Rows,
       Left => Current_C > 1, Right => Current_C < Cols);

   procedure Move(The_Move: Moves) is
      Old_R: Row_Type; Old_C: Col_Type; N: Natural;
   begin
      if not Possible(The_Move) then
     raise Constraint_Error with "attempt to make impossible move";
      else
     -- remember current row and column
     Old_R := Current_R;
     Old_C := Current_C;

     -- move the virtual cursor to a new position
     case The_Move is
       when Up    => Current_R := Current_R - 1;
       when Down  => Current_R := Current_R + 1;
       when Left  => Current_C := Current_C - 1;
       when Right => Current_C := Current_C + 1;
     end case;

     -- swap the tiles on the board
     N := Field(Old_R, Old_C);
     Field(Old_R, Old_C) := Field(Current_R, Current_C);
```

```
      Field(Current_R, Current_C) := N;
       end if;
   end Move;

begin
   declare   -- set field to its basic setting
      N: Positive := 1;
   begin
      for R in Row_Type loop
     for C in Col_Type loop
        if (R /= Current_R) or else (C /= Current_C) then
           Field(R, C) := N;
           N := N + 1;
        else
           Field(R, C) := 0;
        end if;
     end loop;
      end loop;
   end;
end Generic_Puzzle;
```

The main program reads the level from the command line. A larger level implies a more difficult instance. The default level is 10, which is fairly simple. After randomizing the board, the user can move the tiles.

```
with Generic_Puzzle, Ada.Text_IO,
     Ada.Numerics.Discrete_Random, Ada.Command_Line;

procedure Puzzle_15 is

   function Image(N: Natural) return String is
      (if N=0 then "   " elsif N < 10 then " " & Integer'Image(N)
    else Integer'Image(N));

   package Puzzle is new Generic_Puzzle(Rows => 4, Cols => 4, Name => Image);

   package Rnd is new Ada.Numerics.Discrete_Random(Puzzle.Moves);
   Rand_Gen: Rnd.Generator;

   Level: Natural := (if Ada.Command_Line.Argument_Count = 0 then 10
                      else Natural'Value(Ada.Command_Line.Argument(1)));
   Initial_Moves: Natural := (2**(Level/2) + 2**((1+Level)/2))/2;
   Texts: constant array(Puzzle.Moves) of String(1..9) :=
       ("u,U,^,8: ", "d,D,v,2: ", "l,L,<,4: ", "r,R,>,6: ");
   Move_Counter: Natural := 0;
   Command: Character;

 begin
    -- randomize board
    for I in 1 .. Initial_Moves loop
       declare
      M: Puzzle.Moves := Rnd.Random(Rand_Gen);
       begin
      if Puzzle.Possible(M) then
         Puzzle.Move(M);
      end if;
       end;
    end loop;

    -- read command and perform move
    loop
      -- Print board
      for R in Puzzle.Row_Type loop
     for C in Puzzle.Col_Type loop
        Ada.Text_IO.Put(Puzzle.Get_Point(R, C));
     end loop;
     Ada.Text_IO.New_Line;
      end loop;
      Ada.Text_IO.Get(Command);
      begin
     case Command is
        when 'u' | 'U' | '^' | '8' =>
           Ada.Text_IO.Put_Line("Up!"); Puzzle.Move(Puzzle.Up);
        when 'd' | 'D' | 'v' | '2' =>
```

```
            Ada.Text_IO.Put_Line("Down!"); Puzzle.Move(Puzzle.Down);
         when 'l' | 'L' | '<' | '4' =>
            Ada.Text_IO.Put_Line("Left!"); Puzzle.Move(Puzzle.Left);
         when 'r' | 'R' | '>' | '6' =>
            Ada.Text_IO.Put_Line("Right!"); Puzzle.Move(Puzzle.Right);
         when '!' =>
            Ada.Text_IO.Put_Line(Natural'Image(Move_Counter) & " moves!");
            exit;
         when others =>
            raise Constraint_Error with "wrong input";
      end case;
      Move_Counter := Move_Counter + 1;
       exception when Constraint_Error =>
      Ada.Text_IO.Put_Line("Possible Moves and Commands:");
      for M in Puzzle.Moves loop
         if Puzzle.Possible(M) then
            Ada.Text_IO.Put(Texts(M) & Puzzle.Moves'Image(M) & "   ");
         end if;
      end loop;
      Ada.Text_IO.Put_Line("!: Quit");
       end;
    end loop;
end Puzzle_15;
```

Output:

```
>./puzzle_15 4
  1  2  3  4
  5  6  7  8
  9 14 10 11
 13    15 12
8
Up!
  1  2  3  4
  5  6  7  8
  9    10 11
 13 14 15 12
6
Right!
  1  2  3  4
  5  6  7  8
  9 10    11
 13 14 15 12
5
Possible Moves and Commands:
u,U,^,8: UP   d,D,v,2: DOWN   l,L,<,4: LEFT   r,R,>,6: RIGHT   !: Quit
  1  2  3  4
  5  6  7  8
  9 10    11
 13 14 15 12
6
Right!
  1  2  3  4
  5  6  7  8
  9 10 11
 13 14 15 12
2
Down!
  1  2  3  4
  5  6  7  8
  9 10 11 12
 13 14 15
!
 4 moves!
```

For other puzzles, one must just the single line with the package instantiation. E.g., for an 8-puzzle, we would write the following.

```
   package Puzzle is new Generic_Puzzle(Rows => 3, Cols => 3, Name => Image);
```

Amazing Hopper

```
#include <jambo.h>

#define FILATABLA            5
#define COLUMNATABLA         10
#define Imprimelamatriz      Gosub 'Pone la matriz'
#define Imprimelascasillas   Gosub 'Pone las casillas'
#define Imprimeelíndiceen(_X_,_Y_)  Set '_X_,_Y_',  Gosub 'Pone el índice'

Main
   Set break

   Void (casilla, índice, números)
   Link gosub( Crea una casilla, Crea el índice, Crea la matriz de números )

   Cls
   x=4, y=4, Tok sep '""', Gosub 'Imprime escenario'

  /* INICIA EL JUEGO */
   SW = 1, GANADOR = 0
   c=0, cero x=4, cero y=4

   Loop
       Let ( c:=Getch )
       Switch ( c )
           Case 'KRIGHT'  { #( y < 4 ) do{ ++y }, Exit }
           Case 'KDOWN'   { #( x < 4 ) do{ ++x }, Exit }
           Case 'KLEFT'   { #( y > 1 ) do{ --y }, Exit }
           Case 'KUP'     { #( x > 1 ) do{ --x }, Exit }
           Case 'KRETURN' { If ( Gosub 'Chequear si movimiento es válido' )
                               Gosub 'Mover las casillas'
                            End If
                            Exit
                          }
           Case 'KESCAPE' { SW=0 }
       End switch

       Gosub 'Imprime escenario'
       Break if ( Gosub 'Verificar puzzle resuelto' --- Backup to 'GANADOR' )
   Back if 'SW' is not zero

  /* FIN DEL JUEGO */
   If ( GANADOR )
       Locate (18,15), Printnl("LO RESOLVISTE!")
   End If
   Locate (19,1), Prnl
End

Subrutines

/* CHEQUEO DE MOVIMIENTO */

Define ( Verificar puzzle resuelto )
   ret = 0
   Clr all marks
   Tnúmeros=números
   Redim (Tnúmeros,0), N = 0, Let ( N := Length(Tnúmeros) Minus (1))
   i=1
   Iterator ( ++i, Less equal ( i, N ) And( Not(ret) ), \
           Let ( ret := Bit xor(i, [i] Get 'Tnúmeros') ) )
   Clr all marks
   Clear 'Tnúmeros'

Return 'Not (ret); And( Equals(i, Plus one(N)) ) '

Define ( Chequear si movimiento es válido )
Return 'Only one ( Equals (x, cero x), Equals(y, cero y ) )'

Define ( Mover las casillas )
   If ( Equals (y, cero y) )
```

```
      If ( Less (x, cero x) )      // mueve hacia abajo
         Loop for ( i = cero x, #( i >= x ) , --i )
             If ( Greater ( i, 1 ) )
                 [{i} Minus(1), y] Get 'números', [i,y] Put 'números'
             Else
                 [{i} Plus(1), y] Get 'números', [i,y] Put 'números'
             End If
         Next
      Else                         // por defecto: mueve hacia arriba
         Loop for ( i = cero x, #( i <= x ) , ++i )
             If ( Less ( i, 4 ) )
                 [{i} Plus(1), y] Get 'números', [i,y] Put 'números'
             Else
                 [i,y] Get 'números', [{i} Minus(1),y] Put 'números'
             End If
         Next
      End If
      [x,y] Set '0', Put 'números'
      Set 'x', Move to 'cero x'
   Else                            // por defecto: está en la misma fila
      If ( Less ( y, cero y ) )   // mueve hacia la derecha
         Loop for ( i = cero y, #( i >= y ) , --i )
             If ( Greater ( i, 1) )
                 [x, {i} Minus(1)] Get 'números', [x,i] Put 'números'
             Else
                 [x, y] Get 'números', [x, {i} Plus(1)] Put 'números'
             End If
         Next
      Else                         // por defecto: mueve hacia la izquierda
         Loop for ( i = cero y, #( i <= y ) , ++i )
             If ( Less ( i, 4 ) )
                 [x, {i} Plus(1)] Get 'números', [x,i] Put 'números'
             Else
                 [x,i] Get 'números', [x,{i} Minus(1)] Put 'números'
             End If
         Next
      End If
      [x,y] Set '0', Put 'números'
      Set 'y', Move to 'cero y'
   End If
   Clr all marks
Return

/* DESPLIEGUE DE CUADRITOS Y NUMEROS */

Define ( Imprime escenario )
   Imprime las casillas
   Imprime el índice en 'x,y'
   Imprime la matriz
Return

Define ( Pone la matriz )
   i=4, col = COLUMNA TABLA, celda=""
   Clr all marks
   py=1
   Loop
      j=4, fil = FILA TABLA, px=1
      Loop
         Locate 'Plus one(fil), Plus two (col)'
         Printnl( Get if ([px,py] Get 'números' ---Backup to (celda)---, celda, "  ") )
         fil += 3
         --j, ++px
      Back if (j) is not zero
      col += 6, --i, ++py
   Back if (i) is not zero
Return

Define ( Pone las casillas )
   i=4, col = COLUMNA TABLA
   Clr all marks
   Loop
      j=4, fil = FILA TABLA
      Loop
         Set 'fil, col', Gosub 'Pone un cuadrito'
         fil += 3, --j
      Back if (j) is not zero
      col += 6, --i
   Back if (i) is not zero
```

```
Return

Define (Pone un cuadrito, fil, col)
   Locate 'fil, col', Print table 'casilla'
Return

Define ( Pone el índice, fil, col )
   /* 5+(fil-1)*3 fila
      10+(col-1)*6 col */
   Clr all marks
   Locate 'Minus one(fil) Mul by (3) Plus (FILA TABLA), Minus one(col) Mulby(6) Plus(COLUMNA TABLA)'
   Print table 'índice'
Return

/* CONFIGURACION DEL JUEGO */

Define ( Crea la matriz de números )
   Sequence ( 0, 1, 16, números )
   Gosub 'Barajar el array'
   Redim ( números, 4,4 )
Return

/* algoritmo de Fisher-Yates */
Define ( Barajar el array )
  N = 0, Let ( N := Length(números) )
  R = 0, aux = 0
  Loop
     Let (R := Ceil(Rand(N)))
     Let (aux := [R] Get 'números' )
     [N] Get 'números', [R] Put 'números'
     Set 'aux', [N] Put 'números'
     --N
  Back if 'N' is positive
  If ( [16] Get 'números' ---Backup to 'aux'---, Not (Is zero?) )
     [aScan(1,0,números)] Set 'aux', Put 'números'
     [16] Set '0', Put 'números'
  End If
Return

Define ( Crea una casilla )
   Set 'Utf8(Chr(218)),Utf8(Chr(196)),Utf8(Chr(196)),Utf8(Chr(196)),Utf8(Chr(196)),Utf8(Chr(191))', Apnd row
'casilla'
   Set 'Utf8(Chr(179)),"  ","  ","  ","  ",Utf8(Chr(179))', Apnd row 'casilla'
   Set 'Utf8(Chr(192)),Utf8(Chr(196)),Utf8(Chr(196)),Utf8(Chr(196)),Utf8(Chr(196)),Utf8(Chr(217))', Apnd row
'casilla'
Return

Define ( Crea el índice )
   Set 'Utf8(Chr(220)),Utf8(Chr(220)),Utf8(Chr(220)),Utf8(Chr(220)),Utf8(Chr(220)),Utf8(Chr(220))', Apnd row
'índice'
   Set 'Utf8(Chr(219)),"  ","  ","  ","  ",Utf8(Chr(219))', Apnd row 'índice'
   Set 'Utf8(Chr(223)),Utf8(Chr(223)),Utf8(Chr(223)),Utf8(Chr(223)),Utf8(Chr(223)),Utf8(Chr(223))', Apnd row
'índice'
Return
```

Output:

```
$ hopper jm/puzzle.jambo
          ┌────┬────┬────┬────┐
          │ 14 │ 5  │ 3  │ 12 │
          └────┴────┴────┴────┘

          ┌────┬────┬────┬────┐
          │ 13 │ 9  │ 6  │ 11 │
          └────┴────┴────┴────┘

          ┌────┬────┬────┬────┐
          │ 15 │ 10 │ 8  │ 2  │
          └────┴────┴────┴────┘

          ┌────┬────┬────┬─────┐
          │ 4  │ 1  │ 7  │     │
          └────┴────┴────┴─────┘
         ..... (muchos click)
          ┌────┬────┬────┬────┐
          │ 7  │ 10 │ 14 │ 12 │
          └────┴────┴────┴────┘

          ┌────┬────┬────┬────┐
```

```
 ┌──┐ ┌──┐ ┌──┐ ┌──┐
 │15│ │  │ │ 4│ │ 2│
 └──┘ └──┘ └──┘ └──┘
 ┌──┐ ┌──┐ ┌──┐ ┌──┐
 │ 1│ │ 8│ │ 6│ │11│
 └──┘ └──┘ └──┘ └──┘
 ┌──┐ ┌──┐ ┌──┐ ┌──┐
 │ 3│ │13│ │ 5│ │ 9│
 └──┘ └──┘ └──┘ └──┘
....( muy muchos clicks )
 ┌──┐ ┌──┐ ┌──┐ ┌──┐
 │ 1│ │ 2│ │ 3│ │ 4│
 └──┘ └──┘ └──┘ └──┘
 ┌──┐ ┌──┐ ┌──┐ ┌──┐
 │ 5│ │ 6│ │ 7│ │ 8│
 └──┘ └──┘ └──┘ └──┘
 ┌──┐ ┌──┐ ┌──┐ ┌──┐
 │ 9│ │10│ │11│ │12│
 └──┘ └──┘ └──┘ └──┘
 ┌──┐ ┌──┐ ┌──┐ ┌──┐
 │13│ │14│ │15│ │  │
 └──┘ └──┘ └──┘ └──┘

     LO RESOLVISTE!
```

APL

Works with: Dyalog APL version 16.0

```
fpg←{⎕IO←0
    α←4 4
    (sV.<0)∨2≠ρs←α:'invalid shape:'s
    0≠ρρω:'invalid shuffle count:'ω
    d←d,-d←↓2 2ρ3↑1
    e←¯1+ρc←'↑↓↔○'
    b←w←sρw←1⌽⍳×/s
    z←⊃{
        z p←ω
        n←(?ρp)⊃p←(p≡¨(⊂s)⌈p)/p←(d~p)+⊂z
        b[z n]←b[n z]
        -⍨\n z
    }⍣ω⊢(s-1)0
    ⎕←b
    ⍬{
        b≡w:'win'
        0=ρα:⍞←∇ ω
        e=i←c⍳m←⊃α:'quit'
        i>e:⍞←∇ ω⊣⎕←'invalid direction:'m
        n≢⍬⌈n←ω+i⊃d:⍞←∇ ω⊣'out of bounds:'m
        b[ω n]←b[n ω]
        ⎕←(s×0≠ρα)ρb
        (1↓α)∇ n
    }z
}
```

Output:

```
      fpg 10
 1  3  0  4
 5  2  6  8
 9 10  7 12
13 14 11 15
←
 1  0  3  4
 5  2  6  8
 9 10  7 12
13 14 11 15
↓
 1  2  3  4
 5  0  6  8
 9 10  7 12
13 14 11 15
→
```

```
 1  2  3  4
 5  6  0  8
 9 10  7 12
13 14 11 15
↓↓
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14  0 15
→
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
win


      2 5 fpg 2
1 2 3 0 4
6 7 8 9 5
→
1 2 3 4 0
6 7 8 9 5
↓
1 2 3 4 5
6 7 8 9 0
win
```

ARM Assembly

Works with: as version Raspberry Pi

```
/* ARM assembly Raspberry PI  */
/*  program puzzle15.s   */

/*********************************/
/* Constantes                    */
/*********************************/
.equ STDIN,  0     @ Linux input console
.equ STDOUT, 1     @ Linux output console
.equ EXIT,   1     @ Linux syscall
.equ READ,   3     @ Linux syscall
.equ WRITE,  4     @ Linux syscall

.equ IOCTL,     0x36  @ Linux syscall
.equ SIGACTION, 0x43  @ Linux syscall
.equ SYSPOLL,   0xA8  @ Linux syscall

.equ TCGETS,    0x5401
.equ TCSETS,    0x5402
.equ ICANON,    2
.equ ECHO,     10
.equ POLLIN,    1

.equ SIGINT,   2    @ Issued if the user sends an interrupt signal (Ctrl + C)
.equ SIGQUIT,  3    @ Issued if the user sends a quit signal (Ctrl + D)
.equ SIGTERM, 15    @ Software termination signal (sent by kill by default)
.equ SIGTTOU, 22    @

.equ NBBOX,  16
.equ TAILLEBUFFER,   10

/*****************************************/
/* Structures                            */
/*****************************************/
/* structure termios see doc linux*/
    .struct  0
term_c_iflag:                   @ input modes
    .struct  term_c_iflag + 4
term_c_oflag:                   @ output modes
    .struct  term_c_oflag + 4
term_c_cflag:                   @ control modes
    .struct  term_c_cflag + 4
term_c_lflag:                   @ local modes
```

```
    .struct  term_c_lflag + 4
term_c_cc:                      @ special characters
    .struct  term_c_cc + 20     @ see length if necessary
term_fin:

/* structure sigaction see doc linux */
    .struct  0
sa_handler:
    .struct  sa_handler + 4
sa_mask:
    .struct  sa_mask + 4
sa_flags:
    .struct  sa_flags + 4
sa_sigaction:
    .struct  sa_sigaction + 4
sa_fin:

/* structure poll see doc linux */
    .struct  0
poll_fd:                        @   File Descriptor
    .struct  poll_fd + 4
poll_events:                    @  events mask
    .struct  poll_events + 4
poll_revents:                   @ events returned
    .struct  poll_revents + 4
poll_fin:
/*********************************/
/* Initialized data              */
/*********************************/
.data
sMessResult:           .ascii " "
sMessValeur:           .fill 11, 1, ' '             @ size => 11
szCarriageReturn:      .asciz "\n"
szMessGameWin:         .ascii "You win in "
sMessCounter:           .fill 11, 1, ' '            @ size => 11
                       .asciz " move number !!!!\n"
szMessMoveError:       .asciz "Huh... Impossible move !!!!\n"
szMessErreur:          .asciz "Error detected.\n"
szMessSpaces:          .asciz "    "
iGraine:               .int 123456
/*************************************************/
szMessErr: .ascii   "Error code hexa : "
sHexa: .space 9,' '
         .ascii "  decimal :  "
sDeci: .space 15,' '
         .asciz "\n"
szClear:     .byte 0x1B
             .byte 'c'                          @ console clear
             .byte 0
/*********************************/
/* UnInitialized data            */
/*********************************/
.bss
.align 4
iCodeError:     .skip 4
ibox:          .skip 4 * NBBOX                 @ game boxes
iEnd:           .skip 4                        @ 0 loop  1 = end loop
iTouche:        .skip 4                        @ value key pressed
stOldtio:       .skip term_fin                 @ old terminal state
stCurtio:       .skip term_fin                 @ current terminal state
stSigAction:    .skip sa_fin                   @ area signal structure
stSigAction1:   .skip sa_fin
stPoll1:        .skip poll_fin                 @ area poll structure
stPoll2:        .skip poll_fin
/*********************************/
/*  code section                 */
/*********************************/
.text
.global main
main:                               @ entry of program
    mov r0,#0
    ldr r2,iAdribox
    mov r9,#0                       @ move counter
1:                                  @ loop init boxs
    add r1,r0,#1                    @ box value
    str r1,[r2,r0, lsl #2]          @ store value
    add r0,#1                       @ increment counter
    cmp r0,#NBBOX - 2               @ end ?
```

```
    ble 1b
    mov r10,#15                     @ empty box location
    ldr r0,iAdribox
    bl shuffleGame
2:                                  @ loop moves
    ldr r0,iAdribox
    bl displayGame
    //ldr r0,iAdribox
    //bl gameOK                     @ end game ?
    //cmp r0,#1
    //beq 50f
    bl readKey                      @ read key
    cmp r0,#-1
    beq 100f                        @ error or control-c
    mov r1,r0                       @ key
    ldr r0,iAdribox
    bl keyMove
    ldr r0,iAdribox
    bl gameOK                       @ end game ?
    cmp r0,#1
    bne 2b                          @ no -> loop
50:                                 @ win
    mov r0,r9                       @ move counter
    ldr r1,iAdrsMessCounter
    bl conversion10
    ldr r0,iAdrszMessGameWin
    bl affichageMess

100:                                @ standard end of the program
    mov r0, #0                      @ return code
    mov r7, #EXIT                   @ request to exit program
    svc #0                          @ perform the system call

iAdrsMessValeur:          .int sMessValeur
iAdrszCarriageReturn:     .int szCarriageReturn
iAdrsMessResult:          .int sMessResult
iAdribox:                 .int ibox
iAdrszMessGameWin:        .int szMessGameWin
iAdrsMessCounter:         .int sMessCounter
/******************************************************************/
/*     key move                                                   */
/******************************************************************/
/* r0 contains boxs address           */
/* r1 contains key value               */
/* r9 move counter                     */
/* r10 contains location empty box    */
keyMove:
    push {r1-r8,lr}                 @ save  registers
    mov r8,r0
    cmp r1,#0x42                    @ down arrow
    bne 1f
    cmp r10,#4                      @ if r10 < 4   error
    blt 80f
    sub r2,r10,#4                   @ compute location
    b 90f
1:
    cmp r1,#0x41                    @ high arrow
    bne 2f
    cmp r10,#11                     @ if r10 > 11   error
    bgt 80f
    add r2,r10,#4                   @ compute location
    b 90f
2:
    cmp r1,#0x43                    @ right arrow
    bne 3f
    tst r10,#0b11                   @ if r10 = 0,4,8,12   error
    beq 80f
    sub r2,r10,#1                   @ compute location
    b 90f
3:
    cmp r1,#0x44                    @ left arrow
    bne 100f
    and r3,r10,#0b11                @ error if r10 = 3 7 11 and 15
    cmp r3,#3
    beq 80f
    add r2,r10,#1                   @ compute location
    b 90f
```

```
80:                                 @ move error
    ldr r0,iAdriCodeError
    mov r1,#1
    str r1,[r0]
    b 100f
90:                                 @ white box and move box inversion
    ldr r3,[r8,r2,lsl #2]
    str r3,[r8,r10,lsl #2]
    mov r10,r2
    mov r3,#0
    str r3,[r8,r10,lsl #2]
    add r9,#1                       @ increment move counter
100:
    pop {r1-r8,lr}                  @ restaur registers
    bx lr                           @return
iAdriCodeError:             .int iCodeError
/******************************************************************/
/*     shuffle game                                               */
/******************************************************************/
/* r0 contains boxs address           */
shuffleGame:
    push {r1-r6,lr}                   @ save  registers
    mov r1,r0
    mov r0,#4
    bl genereraleas
    lsl r4,r0,#1
    mov r0,r8
1:
    mov r0,#14
    bl genereraleas
    add r6,r0,#1
    mov r0,#14
    bl genereraleas
    add r5,r0,#1
    ldr r2,[r1,r6,lsl #2]
    ldr r3,[r1,r5,lsl #2]
    str r2,[r1,r5,lsl #2]
    str r3,[r1,r6,lsl #2]
    subs r4,#1
    bgt 1b

100:
    pop {r1-r6,lr}                    @ restaur registers
    bx lr                             @return
/******************************************************************/
/*     game Ok ?                                                  */
/******************************************************************/
/* r0 contains boxs address           */
gameOK:
    push {r1-r8,lr}                   @ save  registers
    mov r8,r0
    mov r2,#0
    ldr r3,[r8,r2,lsl #2]
    add r2,#1
1:
    ldr r4,[r8,r2,lsl #2]
    cmp r4,r3
    movlt r0,#0                       @ game mot Ok
    blt 100f
    mov r3,r4
    add r2,#1
    cmp r2,#NBBOX -2
    ble 1b
    mov r0,#1                         @ game Ok

100:
    pop {r1-r8,lr}                    @ restaur registers
    bx lr                             @return
/******************************************************************/
/*     display game                                               */
/******************************************************************/
/* r0 contains boxs address           */
displayGame:
    push {r1-r5,lr}                   @ save  registers
    @ clear !
    mov r4,r0
    ldr r0,iAdrszClear
    bl affichageMess
```

```
    mov r2,#0
    ldr r1,iAdrsMessValeur
1:
    ldr r0,[r4,r2,lsl #2]
    cmp r0,#0
    ldreq r0,iSpaces                    @ store spaces
    streq r0,[r1]
    beq 2f
    bl conversion10                     @ call conversion decimal
    mov r0,#0
    strb r0,[r1,#3]                     @ zéro final
2:

    ldr r0,iAdrsMessResult
    bl affichageMess                    @ display message
    add r0,r2,#1
    tst r0,#0b11
    bne 3f
    ldr r0,iAdrszCarriageReturn
    bl affichageMess                    @ display message
3:
    add r2,#1
    cmp r2,#NBBOX - 1
    ble 1b
    ldr r0,iAdrszCarriageReturn
    bl affichageMess                    @ display line return
    ldr r0,iAdriCodeError               @ error detected ?
    ldr r1,[r0]
    cmp r1,#0
    beq 100f
    mov r1,#0                           @ raz error code
    str r1,[r0]
    ldr r0,iAdrszMessMoveError          @ display error message
    bl affichageMess
100:
    pop {r1-r5,lr}                      @ restaur registers
    bx lr                               @return
iSpaces:                        .int 0x00202020       @ spaces
iAdrszClear:                    .int szClear
iAdrszMessMoveError:            .int szMessMoveError
/******************************************************************/
/*     display text with size calculation                         */
/******************************************************************/
/* r0 contains the address of the message */
affichageMess:
    push {r0,r1,r2,r7,lr}                           @ save  registres
    mov r2,#0                                       @ counter length
1:                                                  @ loop length calculation
    ldrb r1,[r0,r2]                                 @ read octet start position + index
    cmp r1,#0                                       @ if 0 its over
    addne r2,r2,#1                                  @ else add 1 in the length
    bne 1b                                          @ and loop
                                                    @ so here r2 contains the length of the message
    mov r1,r0                                       @ address message in r1
    mov r0,#STDOUT                                  @ code to write to the standard output Linux
    mov r7, #WRITE                                  @ code call system "write"
    svc #0                                          @ call systeme
    pop {r0,r1,r2,r7,lr}                            @ restaur des  2 registres */
    bx lr                                           @ return
/******************************************************************/
/*     Converting a register to a decimal unsigned                */
/******************************************************************/
/* r0 contains value and r1 address area   */
/* r0 return size of result (no zero final in area) */
/* area size => 11 bytes          */
.equ LGZONECAL,   10
conversion10:
    push {r1-r4,lr}                                 @ save registers
    mov r3,r1
    mov r2,#LGZONECAL
1:                                                  @ start loop
    bl divisionpar10U                               @ unsigned  r0 <- dividende. quotient ->r0 reste -> r1
    add r1,#48                                      @ digit
    strb r1,[r3,r2]                                 @ store digit on area
    cmp r0,#0                                       @ stop if quotient = 0
    subne r2,#1                                     @ else previous position
    bne 1b                                          @ and loop
                                                    @ and move digit from left of area
```

```
    mov r4,#0
2:
    ldrb r1,[r3,r2]
    strb r1,[r3,r4]
    add r2,#1
    add r4,#1
    cmp r2,#LGZONECAL
    ble 2b
                                                @ and move spaces in end on area
    mov r0,r4                                   @ result length
    mov r1,#' '                                 @ space
3:
    strb r1,[r3,r4]                             @ store space in area
    add r4,#1                                   @ next position
    cmp r4,#LGZONECAL
    ble 3b                                      @ loop if r4 <= area size

100:
    pop {r1-r4,lr}                              @ restaur registres
    bx lr                                       @return

/***************************************************/
/*   division par 10   unsigned                    */
/***************************************************/
/* r0 dividende   */
/* r0 quotient    */
/* r1 remainder   */
divisionpar10U:
    push {r2,r3,r4, lr}
    mov r4,r0                                    @ save value
    ldr r3,iMagicNumber                          @ r3 <- magic_number    raspberry 1 2
    umull r1, r2, r3, r0                         @ r1<- Lower32Bits(r1*r0) r2<- Upper32Bits(r1*r0)
    mov r0, r2, LSR #3                           @ r2 <- r2 >> shift 3
    add r2,r0,r0, lsl #2                         @ r2 <- r0 * 5
    sub r1,r4,r2, lsl #1                         @ r1 <- r4 - (r2 * 2)  = r4 - (r0 * 10)
    pop {r2,r3,r4,lr}
    bx lr                                        @ leave function
iMagicNumber:   .int 0xCCCCCCCD
/***************************************************/
/*   Generation random number                  */
/***************************************************/
/* r0 contains limit  */
genereraleas:
    push {r1-r4,lr}                              @ save registers
    ldr r4,iAdriGraine
    ldr r2,[r4]
    ldr r3,iNbDep1
    mul r2,r3,r2
    ldr r3,iNbDep1
    add r2,r2,r3
    str r2,[r4]                                  @ maj de la graine pour l appel suivant
    cmp r0,#0
    beq 100f
    mov r1,r0                                    @ divisor
    mov r0,r2                                    @ dividende
    bl division
    mov r0,r3                                    @ résult = remainder

100:                                             @ end function
    pop {r1-r4,lr}                               @ restaur registers
    bx lr                                        @ return
/***************************************************/
iAdriGraine: .int iGraine
iNbDep1: .int 0x343FD
iNbDep2: .int 0x269EC3
/***************************************************/
/* integer division unsigned                     */
/***************************************************/
division:
    /* r0 contains dividend */
    /* r1 contains divisor */
    /* r2 returns quotient */
    /* r3 returns remainder */
    push {r4, lr}
    mov r2, #0                                   @ init quotient
    mov r3, #0                                   @ init remainder
    mov r4, #32                                  @ init counter bits
    b 2f
```

```
1:                                                  @ loop
    movs r0, r0, LSL #1                             @ r0 <- r0 << 1 updating cpsr (sets C if 31st bit of r0 was
1)
    adc r3, r3, r3                                  @ r3 <- r3 + r3 + C. This is equivalent to r3 ? (r3 << 1) +
C
    cmp r3, r1                                      @ compute r3 - r1 and update cpsr
    subhs r3, r3, r1                                @ if r3 >= r1 (C=1) then r3 <- r3 - r1
    adc r2, r2, r2                                  @ r2 <- r2 + r2 + C. This is equivalent to r2 <- (r2 << 1)
+ C
2:
    subs r4, r4, #1                                 @ r4 <- r4 - 1
    bpl 1b                                          @ if r4 >= 0 (N=0) then loop
    pop {r4, lr}
    bx lr
/*************************************************/
/* read touch                                    */
/*************************************************/
readKey:
    push {r1-r7,lr}
    mov r5,#0
    /* read terminal state */
    mov r0,#STDIN                                   @ input console
    mov r1,#TCGETS
    ldr r2,iAdrstOldtio
    mov r7, #IOCTL                                  @ call system Linux
    svc #0
    cmp r0,#0                                       @ error ?
    beq 1f
    ldr r1,iAdrszMessErreur                         @ error message
    bl   displayError
    mov r0,#-1
    b 100f
1:
    adr r0,sighandler                               @ adresse routine traitement signal
    ldr r1,iAdrstSigAction                          @ adresse structure sigaction
    str r0,[r1,#sa_handler]                         @ maj handler
    mov r0,#SIGINT                                  @ signal type
    ldr r1,iAdrstSigAction
    mov r2,#0                                       @ NULL
    mov r7, #SIGACTION                              @ call system
    svc #0
    cmp r0,#0                                       @ error ?
    bne 97f
    mov r0,#SIGQUIT
    ldr r1,iAdrstSigAction
    mov r2,#0                                       @ NULL
    mov r7, #SIGACTION                              @ call system
    svc #0
    cmp r0,#0                                       @ error ?
    bne 97f
    mov r0,#SIGTERM
    ldr r1,iAdrstSigAction
    mov r2,#0                                       @ NULL
    mov r7, #SIGACTION                              @ appel systeme
    svc #0
    cmp r0,#0
    bne 97f
    @
    adr r0,iSIG_IGN                                 @ address signal ignore function
    ldr r1,iAdrstSigAction1
    str r0,[r1,#sa_handler]
    mov r0,#SIGTTOU                                 @invalidate other process signal
    ldr r1,iAdrstSigAction1
    mov r2,#0                                       @ NULL
    mov r7,#SIGACTION                               @ call system
    svc #0
    cmp r0,#0
    bne 97f
    @
    /* read terminal current state  */
    mov r0,#STDIN
    mov r1,#TCGETS
    ldr r2,iAdrstCurtio                             @ address current termio
    mov r7,#IOCTL                                   @ call systeme
    svc #0
    cmp r0,#0                                       @ error ?
    bne 97f
    mov r2,#ICANON | ECHO                           @ no key pressed echo on display
```

```
    mvn r2,r2                                   @ and one key
    ldr r1,iAdrstCurtio
    ldr r3,[r1,#term_c_lflag]
    and r3,r2                                   @ add flags
    str r3,[r1,#term_c_lflag]                   @ and store
    mov r0,#STDIN                               @ maj terminal current state
    mov r1,#TCSETS
    ldr r2,iAdrstCurtio
    mov r7, #IOCTL                              @ call system
    svc #0
    cmp r0,#0
    bne 97f
    @
2:                                              @ loop waiting key
    ldr r0,iAdriEnd                             @ if signal ctrl-c  -> end
    ldr r0,[r0]
    cmp r0,#0
    movne r5,#-1
    bne 98f
    ldr r0,iAdrstPoll1                           @ address structure poll
    mov r1,#STDIN
    str r1,[r0,#poll_fd]                         @ maj FD
    mov r1,#POLLIN                               @ action code
    str r1,[r0,#poll_events]
    mov r1,#1                                    @ items number structure poll
    mov r2,#0                                    @ timeout = 0
    mov r7,#SYSPOLL                              @ call system POLL
    svc #0
    cmp r0,#0                                    @ key pressed ?
    ble 2b                                       @ no key pressed -> loop
                                                 @ read key
    mov r0,#STDIN                                @ File Descriptor
    ldr r1,iAdriTouche                           @ buffer address
    mov r2,#TAILLEBUFFER                         @ buffer size
    mov r7,#READ                                 @ read key
    svc #0
    cmp r0,#0                                    @ error ?
    bgt 98f

97:                                              @ error detected
    ldr r1,iAdrszMessErreur                      @ error message
    bl   displayError
    mov r5,#-1
98:                                              @ end then restaur begin state terminal
    mov r0,#STDIN
    mov r1,#TCSETS
    ldr r2,iAdrstOldtio
    mov r7,#IOCTL                                @ call system
    svc #0
    cmp r0,#0
    beq 99f                                      @ restaur ok
    ldr r1,iAdrszMessErreur                      @ error message
    bl   displayError
    mov r0,#-1
    b 100f
99:
    cmp r5,#0                                    @ error or control-c
    ldreq r2,iAdriTouche                         @ key address
    ldreqb r0,[r2,#2]                            @ return key byte
    movne r0,r5                                  @ or error
100:
    pop {r1-r7, lr}
    bx lr
iSIG_IGN:                .int 1
iAdriEnd:                .int iEnd
iAdrstPoll1:             .int stPoll1
iAdriTouche:             .int iTouche
iAdrstOldtio:            .int stOldtio
iAdrstCurtio:            .int stCurtio
iAdrstSigAction:         .int stSigAction
iAdrstSigAction1:        .int stSigAction1
iAdrszMessErreur :       .int szMessErreur
/******************************************************************/
/*     traitement du signal                                       */
/******************************************************************/
sighandler:
    push {r0,r1}
    ldr r0,iAdriEnd
```

```
    mov r1,#1                @ maj zone end
    str r1,[r0]
    pop {r0,r1}
    bx lr
/*************************************************/
/*   display error message                       */
/*************************************************/
/* r0 contains error code  r1 : message address */
displayError:
    push {r0-r2,lr}                         @ save registers
    mov r2,r0                               @ save error code
    mov r0,r1
    bl affichageMess
    mov r0,r2                               @ error code
    ldr r1,iAdrsHexa
    bl conversion16                         @ conversion hexa
    mov r0,r2                               @ error code
    ldr r1,iAdrsDeci                        @ result address
    bl conversion10                         @ conversion decimale
    ldr r0,iAdrszMessErr                    @ display error message
    bl affichageMess
100:
    pop {r0-r2,lr}                          @ restaur registers
    bx lr                                   @ return
iAdrszMessErr:                .int szMessErr
iAdrsHexa:                    .int sHexa
iAdrsDeci:                    .int sDeci
/*****************************************************************/
/*     Converting a register to hexadecimal                      */
/*****************************************************************/
/* r0 contains value and r1 address area   */
conversion16:
    push {r1-r4,lr}                                     @ save registers
    mov r2,#28                                          @ start bit position
    mov r4,#0xF0000000                                  @ mask
    mov r3,r0                                           @ save entry value
1:                                                      @ start loop
    and r0,r3,r4                                        @value register and mask
    lsr r0,r2                                           @ move right
    cmp r0,#10                                          @ compare value
    addlt r0,#48                                        @ <10  ->digit
    addge r0,#55                                        @ >10  ->letter A-F
    strb r0,[r1],#1                                     @ store digit on area and + 1 in area address
    lsr r4,#4                                           @ shift mask 4 positions
    subs r2,#4                                          @  counter bits - 4 <= zero  ?
    bge 1b                                              @  no -> loop

100:
    pop {r1-r4,lr}                                      @ restaur registers
    bx lr                                               @return
```

Arturo

```
;; ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
;; ===>> ~~ Game's functions ~~ <<===
;; --->> ~~ Init functions ~~ <<---

;; This is a solved sample that is used to
;; init and finish the game
solvedTable: @[ "  1  " "  2  " "  3  " "  4  "
                "  5  " "  6  " "  7  " "  8  "
                "  9  " "  10 " "  11 " "  12 "
                "  13 " "  14 " "  15 " "     " ]

;; Use this once in :game's init, to get a player position
;; Q: Why use it once?
;; A: This algorithm is slower than just get a stored varible
;; yet this searches for a string for every value from :game
getPlayerPosition: $[table :block][
    return index table "     "
]

;; This is the object that represents the game
;; 'table » The sample table to generate the game
```

```
define :game [
    table :block
][

    init: [
        ; checks if 'table has 16 elements
        ensure [16 = size this\table]

        ;; The game's table itself
        this\table: (shuffle this\table) ; creates a random game
        ;; The current movement. When less, better is your punctuation
        this\movements: 0
        ;; The current 'playerPosition in table
        ;; Used to evaluate if certain movement is possible or not
        this\playerPosition: getPlayerPosition this\table
        ;; Defines it the gameLoop still running
        this\running?: true
    ]

    ;; A builtin print function that simplifies the use
    print: [
        render {
            Movements: |this\movements|, Position: |this\playerPosition|
            *-----*-----*-----*-----*
             |this\table\0| |this\table\1| |this\table\2| |this\table\3|
            *-----*-----*-----*-----*
             |this\table\4| |this\table\5| |this\table\6| |this\table\7|
            *-----*-----*-----*-----*
             |this\table\8| |this\table\9| |this\table\10| |this\table\11|
            *-----*-----*-----*-----*
             |this\table\12| |this\table\13| |this\table\14| |this\table\15|
            *-----*-----*-----*-----*
        }
    ]

    ;; Compares the internal's 'table with another :block
    compare: [
        if this\table = that
            -> return true
    ]

]

;; These are the commands used internally on game
;; To avoid ambiguity, User's input'll to be translated to this
gameActions: ['up, 'left, 'down, 'right, 'quit]


;; ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
;; -->> Print funnctions <<---

;; A template for print instructions
printInstructions: [
    color #cyan "Type (WASD) to move and (Q) to quit."
]

;; A template for print input warning
;; 'input: the wrong input itself that will be printed
printWrongInput: $[inp :string][
    print color #red
    ~"Wrong input: '|inp|'"
]

;; A template for print input warning
;; 'action: could be 'up, 'down, 'left or 'right
printWrongMovement: $[action :literal][
    print color #red
    ~"Wrong movement. Can't go |action|"
]


;; ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
;; -->> Validators/Checkers functions <<---

;; Checks if a 'input is in 'gameActions
;; Valids for: 'up, 'down, 'left, 'right and 'quit
validInput?: $[inp :any][
    return (in? inp gameActions)
```

```
]

;; Checks if the current movement tried is possible
;; 'game » is the current game
;; 'movement » must be in 'gameActions, but can't be 'quit
validMovement?: $[
    game     :game
    movement :literal
][
    pos: game\playerPosition
    case [movement]
        when? [='up]
            -> return (not? in? pos [0..3])
        when? [='down]
            -> return (not? in? pos [12..15])
        when? [='left]
            -> return (not? in? pos [0 4 8 12])
        when? [='right]
            -> return (not? in? pos [3 7 11 15])
    else
        -> return false
]


;; ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
;; --->> Action functions <<---

;; Gets user input from terminal
;; returning a :literal from 'gameActions
;; Raises: In case of wrong input,
;; will be returned the same input as a :string
parseInput: $[inp :string][
    lowerInp: lower inp
    case [lowerInp]
        when? ["w"] -> return 'up
        when? ["a"] -> return 'left
        when? ["s"] -> return 'down
        when? ["d"] -> return 'right
        when? ["q"] -> return 'quit
    else -> return inp
]

;; Moves the player in Game's Table
;; Note that this's a unsafe function,
;; use 'validMovement? to check a 'movement given a game,
;; and then use this
movePlayer: $[
    game     :game
    movement :literal
][

    position: game\playerPosition

    updateGame: $[
        game             :game
        playerPosition   :integer
        relativePosition :integer
    ][
        try [

            ; 'otherPosition is the real index of the 'relativePosition
            otherPosition: + playerPosition relativePosition

            ; -- Updates the table, swaping the positions
            temp: game\table\[playerPosition]
            game\table\[playerPosition]: game\table\[otherPosition]
            game\table\[otherPosition]: temp

            ; -- Updates player's status
            game\playerPosition: otherPosition
            game\movements: inc game\movements
        ] else -> panic "'movement didn't checked."
    ]

    case [movement]
    when? [='up]
        -> (updateGame game position (neg 4))
    when? [='down]
```

```
        -> (updateGame game position (4))
    when? [='left]
        -> (updateGame game position (neg 1))
    when? [='right]
        -> (updateGame game position (1))
    else -> panic "'movement didn't checked."

]

endGame: $[
    message :string
][
    print message
    exit
]


;; ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
;; --->> Run function <<---

;; Inits ans runs the game
;; 'sampleTable must be already solved
runGame: $[sampleTable :block][
    game: to :game [sampleTable]

    while [game\running?] [
        print game
        print printInstructions
        command: parseInput input ">> "

        if command = 'quit
            -> endGame "Exiting game..."

        validInp: validInput? command
        if? validInp [
            validMov: validMovement? game command
            (validMov)?
                -> movePlayer game command
                -> printWrongMovement command
        ] else
            -> printWrongInput command

        if sampleTable = game
            -> endGame "Congratulations! You won!"
        print ""
    ]
]


runGame solvedTable
```

Astro

```
type Puzzle(var items: {}, var position: -1)

fun mainframe(puz):
    let d = puz.items
    print('+-----+-----+-----+-----+')
    print(d[1], d[2], d[3], d[4], first: '|', sep: '|', last: '|')
    print('+-----+-----+-----+-----+')
    print(d[5], d[6], d[7], d[8], first: '|', sep: '|', last: '|')
    print('+-----+-----+-----+-----+')
    print(d[9], d[10], d[11], d[12], first: '|', sep: '|', last: '|')
    print('+-----+-----+-----+-----+')
    print(d[13], d[14], d[15], d[16], first: '|', sep: '|', last: '|')
    print('+-----+-----+-----+-----+')

fun format(puz, ch):
    match ch.trim().length:
        1 => '  $ch  '
        2 => '  $ch '
        0 => '     '

fun change(puz, to):
```

```
    let fro = puz.position
    for a, b in puz.items where b == puz.format(str i):
        to = a
        break

    swap(puz.items[fro], :[to])
    puz.position = to;

fun buildboard(puz, difficulty):
    for i in 1..16:
        puz.items[i] = puz.format(str i)

    var tmp = a
    for a, b in puz.items where b == '  16 ':
        puz.items[a] = '     '
            tmp = a
            break

    puz.position = tmp
    let diff = match difficulty:
        0 => 10
        1 => 50
        _ => 100

    for i in 1..diff:
        let lst = puz.validmoves()
        let lst1 = []
        for j in lst:
            lst1.push! j.trim().int()
        puz.change(lst1[random(1, lst1.length - 1)])

fun validmoves(puz):
    match puz.position:
        6 | 7 | 10 | 11 =>
            puz.items[pos - 4], :[pos - 1], :[pos + 1], :[pos + 4]
        5 | 9 =>
            puz.items[pos - 4], :[pos + 4], :[pos + 1]
        8 | 12 =>
            puz.items[pos - 4], :[pos + 4], :[pos - 1]
        2 | 3 =>
            puz.items[pos - 1], :[pos + 1], :[pos + 4]
        14 | 15 =>
            puz.items[pos - 1], :[pos + 1], :[pos - 4]
        1 =>
            puz.items[pos + 1], :[pos + 4]
        4 =>
            puz.items[pos - 1], :[pos + 4]
        13 =>
            puz.items[pos + 1], :[pos - 4]
        16 =>
            puz.items[pos - 1], :[pos - 4]

fun mainframe(puz):
    var flag = false
    for a, b in puz.items:
        if b == '     ':
            pass
        else:
            flag = (a == b.trim().int())
    ..
    return flag

let game = Puzzle()
game.buildboard(
    int(input('Enter the difficulty : 0 1 2\n2 => highest 0=> lowest\n'))
)
game.mainframe()

print 'Enter 0 to exit'

loop:
    print 'Hello user:\nTo change the position just enter the no. near it'

    var lst = game.validmoves()
    var lst1 = []
    for i in lst:
        lst1.push! i.trim().int()
        print(i.strip(), '\t', last: '')
```

```
    print()

    let value = int(input())
    if value == 0:
        break
    elif x not in lst1:
        print('Wrong move')
    else:
        game.change(x)

    game.mainframe()
    if g.gameover():
        print 'You WON'
        break
```

AutoHotkey

```
Size := 20
Grid := [], Deltas := ["-1,0","1,0","0,-1","0,1"], Width := Size * 2.5
Gui, font, S%Size%
Gui, add, text, y1
loop, 4
{
    Row := A_Index
    loop, 4
    {
        Col := A_Index
        Gui, add, button, % (Col=1 ? "xs y+1" : "x+1 yp") " v" Row "_" Col " w" Width " gButton -TabStop", %
Grid[Row,Col] := Col + (Row-1)*4 ; 1-16
    }
}
GuiControl, Hide, % Row "_" Col ; 4_4
Gui, add, Button, % "xs gShuffle w" 4 * Width + 3, Shuffle
Gui, show,, 15 Puzzle
return
;------------------------------
GuiClose:
ExitApp
return
;------------------------------
Shuffle:
Shuffle := true
loop, 1000
{
    Random, Rnd, 1,4
    Move(StrSplit(Deltas[Rnd], ",").1, StrSplit(Deltas[Rnd], ",").2)
}
Shuffle := false
return
;------------------------------
Button:
buttonRow := SubStr(A_GuiControl, 1, 1), ButtonCol := SubStr(A_GuiControl, 3, 1)
if Abs(buttonRow-Row) > 1 || Abs(ButtonCol-Col) > 1 || Abs(buttonRow-Row) = Abs(ButtonCol-Col)
    return
Move(buttonRow-Row, ButtonCol-Col)
return
;------------------------------
#IfWinActive, 15 Puzzle
;------------------------------
Down::
Move(-1, 0)
return
;------------------------------
Up::
Move(1, 0)
return
;------------------------------
Right::
Move(0, -1)
return
;------------------------------
Left::
Move(0, 1)
```

```
return
;------------------------------
#IfWinActive
;------------------------------
Move(deltaRow, deltaCol){
    global
    if (Row+deltaRow=0) || (Row+deltaRow=5) || (Col+deltaCol=0) || (Col+deltaCol=5)
        return
    GuiControl, Hide, % Row+deltaRow "_" Col+deltaCol
    GuiControl, Show, % Row "_" Col
    GuiControl,, %Row%_%Col%, % Grid[Row+deltaRow, Col+deltaCol]
    Grid[Row, Col] := Grid[Row+deltaRow, Col+deltaCol]
    Grid[Row+=deltaRow, Col+=deltaCol] := 16
    if Shuffle
        return
    gridCont := ""
    for m, obj in grid
        for n, val in obj
            gridCont .= val ","
    if (Trim(gridCont, ",") = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16")
        MsgBox, 262208, 15 Puzzle, You solved 15 Puzzle
}
```

BASIC

Commodore BASIC

```
10 REM 15-PUZZLE GAME
20 REM COMMODORE BASIC 2.0
30 REM ******************************
40 GOSUB 400 : REM INTRO AND LEVEL
50 GOSUB 510 : REM SETUP BOARD
60 GOSUB 210 : REM PRINT PUZZLE
70 PRINT "TO MOVE A PIECE, ENTER ITS NUMBER:"
80 INPUT X
90 GOSUB 760 : REM CHECK IF MOVE IS VALID
100 IF MV=0 THEN PRINT "WRONG MOVE" : GOSUB 1130 : GOTO 60
110 D(Z)=X : D(Y)=0
120 GOSUB 210 : REM PRINT PUZZLE
130 GOSUB 1030: REM CHECK IF PUZZLE COMPLETE
140 IF PC THEN 160
150 GOTO 70
160 PRINT"YOU WON!"
170 END
180 REM
190 REM ******************************
200 REM PRINT/DRAW THE PUZZLE
210 FOR P=1 TO 16
220   IF D(P)=0 THEN D$(P)="     " : GOTO 260
230   S$=STR$(D(P))
240   N=LEN(S$)
250   D$(P) = LEFT$("   ",3-N)+S$+"  "
260 NEXT
270 PRINT "+-----+-----+-----+-----+"
280 PRINT "!"D$(1)"!"D$(2)"!"D$(3)"!"D$(4)"!"
290 PRINT "+-----+-----+-----+-----+"
300 PRINT "!"D$(5)"!"D$(6)"!"D$(7)"!"D$(8)"!"
310 PRINT "+-----+-----+-----+-----+"
320 PRINT "!"D$(9)"!"D$(10)"!"D$(11)"!"D$(12)"!"
330 PRINT "+-----+-----+-----+-----+"
340 PRINT "!"D$(13)"!"D$(14)"!"D$(15)"!"D$(16)"!"
350 PRINT "+-----+-----+-----+-----+"
360 RETURN
370 REM
380 REM ******************************
390 REM INTRO AND LEVEL OF DIFFICULTY
400 PRINT CHR$(147)
410 DIM SH(3) : SH(1)=10 : SH(2)=50 : SH(3)=100
420 PRINT "15 PUZZLE GAME FOR COMMODORE BASIC 2.0" : PRINT : PRINT
430 PRINT "PLEASE ENTER LEVEL OF DIFFICULTY,"
440 PRINT "1(EASY), 2(MEDIUM) OR 3(HARD):";
450 INPUT V
```

```
460 IF V<1 OR V>3 THEN 440
470 RETURN
480 REM
490 REM ******************************
500 REM BUILD THE BOARD
510 DIM D(16) : DIM D$(16) : REM BOARD PIECES
520 REM SET PIECES IN CORRECT ORDER FIRST
530 FOR P=1 TO 15
540   D(P) = P
550 NEXT
560 D(16) = 0 : REM 0 = EMPTY PIECE/SLOT
570 Z=16      : REM Z = EMPTY POSITION
580 PRINT: PRINT "SHUFFLING PIECES";
590 FOR N=1 TO SH(V)
600   PRINT".";
610   X = INT(RND(0)*4)+1
620   IF X=1 THEN R=Z-4
630   IF X=2 THEN R=Z+4
640   IF (X=3) AND (INT((Z-1)/4)<>(Z-1)/4) THEN R=Z-1
650   IF (X=4) AND (INT(Z/4)<>Z/4) THEN R=Z+1
660   IF R<1 OR R>16 THEN 610
670   D(Z)=D(R)
680   Z=R
690   D(Z)=0
700 NEXT
710 PRINT CHR$(147)
720 RETURN
730 REM
740 REM ******************************
750 REM CHECK IF MOVE IS VALID
760 MV = 0
770 IF X<1 OR X>15 THEN RETURN
780 REM FIND POSITION OF PIECE X AND OF EMPTY PIECE
790 AX=X
800 GOSUB 940 : REM FIND POSITION OF PIECE AX
810 Y=P
820 AX=0
830 GOSUB 940 : REM FIND POSITION OF PIECE AX
840 Z=P
850 REM CHECK IF EMPTY PIECE IS ABOVE, BELOW, LEFT OR RIGHT TO PIECE X
860 IF Y-4=Z THEN MV=1 : RETURN
870 IF Y+4=Z THEN MV=1 : RETURN
880 IF (Y-1=Z) AND (INT(Z/4)<>Z/4) THEN MV=1 : RETURN
890 IF (Y+1=Z) AND (INT(Y/4)<>Y/4) THEN MV=1 : RETURN
900 RETURN
910 REM
920 REM ******************************
930 REM FIND POSITION OF PIECE AX
940 P=1
950 IF D(P)=AX THEN 990
960   P=P+1
970   IF P>16 THEN PRINT "UH OH!" : STOP
980 GOTO 950
990 RETURN
1000 REM
1010 REM ******************************
1020 REM CHECK IF PUZZLE IS COMPLETE / GAME OVER
1030 PC = 0
1040 P=1
1050 IF (P>=16) OR (D(P)<>P) THEN 1080
1060   P=P+1
1070 GOTO 1050
1080 IF P=16 THEN PC=1
1090 RETURN
1100 REM
1110 REM ******************************
1120 REM A SMALL DELAY
1130 FOR T=0 TO 400
1140 NEXT
1150 RETURN
```

BBC BASIC

Works with: BBC BASIC for Windows
Works with: ARM BBC BASIC

Works with: Brandy BASIC version Matrix Brandy

```
      IF INKEY(-256)=77 OR (INKEY(-256) AND &F0)=&A0 THEN MODE 1: COLOUR 0: COLOUR 143: *FX4,1

      SIZE=4 : DIFFICULTY=3

      MAX=SIZE * SIZE - 1
      DIM Board(MAX)
      FOR I%=1 TO MAX : Board(I% - 1)=I% : NEXT
      Gap=MAX
      WHILE N% < DIFFICULTY ^ 2 PROCSlide(RND(4)) : ENDWHILE : REM Shuffle
      N%=0

      @%=2 + LOG(MAX + 1)
      PROCShowAndTest
      WHILE NOT Solved
        PRINT "Use arrow keys to move the gap around. Moves taken: ";N%
        PROCSlide(GET - 135)
        PROCShowAndTest
      ENDWHILE
      PRINT "Solved after ";N% LEFT$(" moves", 6 + (N% = 1)) "."
      END

      DEF PROCSlide(dir%)
      NewGap=Gap
      CASE dir% OF
        WHEN 1: IF Gap MOD SIZE > 0        NewGap=Gap - 1    : N%+=1 : REM Left
        WHEN 2: IF Gap MOD SIZE < SIZE - 1 NewGap=Gap + 1    : N%+=1 : REM Right
        WHEN 3: IF Gap < MAX - SIZE + 1    NewGap=Gap + SIZE : N%+=1 : REM Down
        WHEN 4: IF Gap > SIZE - 1          NewGap=Gap - SIZE : N%+=1 : REM Up
      ENDCASE
      SWAP Board(Gap), Board(NewGap)
      Gap=NewGap
      ENDPROC

      DEF PROCShowAndTest
      CLS
      Solved=TRUE
      FOR I%=0 TO MAX
        COLOUR 12 : COLOUR 135
        IF I% = Gap COLOUR 1 : COLOUR 129
        IF I% MOD SIZE = SIZE - 1 PRINT Board(I%) ELSE PRINT Board(I%),;
        IF Solved IF I% < MAX - 1 IF Board(I%) > Board(I% + 1) OR I% = Gap Solved=FALSE
      NEXT
      COLOUR 0 : COLOUR 143
      PRINT
      ENDPROC
```

BQN

Translation of: APL

```
_while_ ← {𝔽⍟𝔾∘𝔽_𝕣_𝔾∘𝔽⍟𝔾𝕩}
FPG←{
  𝕊𝕩: 4‿4𝕊𝕩;
  (∧´𝕨<0)∨2≠≠𝕨 ? •Out "Invalid shape: "∾•Fmt 𝕨;
  0≠=𝕩 ? •Out "Invalid shuffle count: "∾•Fmt 𝕩;
  s𝕊𝕩:
  d←⟨1‿0∘¯1‿0∘0‿1∘0‿¯1⟩ # Directions
  w←𝕨↩1⌽↕×´𝕨 # Solved grid
  b←w        # Board
  z←⊑{
    z‿p←𝕩
    p↩(⊢≡s⊸|)¨⊸/(<z)+d(¬∘∊/⊣)p # filter out invalid
    n←(•rand.Range ≠p)⊑p
    b⌽◎(z‿n⊸⊑)↩ # switch places
    -`n‿z
  }⍟𝕩 ⟨𝕨-1,⟨0⟩⟩
  {
    𝕊:
    b≡w ? •Show b, •Out "You win", 0;
    •Show b
    inp←⊑{
```

```
      Check 𝕩:
      •Out "Enter move: "
      x←•GetLine@
      i←⊑"↑↓↔q"⊐x
      {
        i=4 ? i; # quit
        i>4 ? •Out "Invalid direction: "∾x, Check x;
        (⊢=≠⊸|)z+i⊏d ? •Out "Out of bounds: "∾x, Check x;
        i
      }
    } @
    {
      𝕩=4 ? •Out "Quitting", 0;
      mv←z+𝕩⊏d
      b⌽◎(mv⌾(z⊸⊏)↩)
      z↩mv
      1
    } inp
  } _while_ ⊢ 1
  @
}
```

```
   )ex 15_puzzle.bqn
   FPG 10
┌─
╵  1  2  0  3
   5  6  7  4
   9 10 11  8
  13 14 15 12
              ┘
Enter move:
a
Invalid direction: a
Enter move:
↑
┌─
╵  1  2  7  3
   5  6  0  4
   9 10 11  8
  13 14 15 12
              ┘
Enter move:
↓
┌─
╵  1  2  0  3
   5  6  7  4
   9 10 11  8
  13 14 15 12
              ┘
Enter move:
↓
Out of bounds: ↓
...
```

C

C89, 22 lines version

The task, as you can see, can be resolved in 22 lines of no more than 80 characters. Of course, the source code in C is not very readable. The second example works exactly the same way, but it was written in much more human readable way. The program also works correctly for non-standard number of rows and/or columns.

```c
/* RosettaCode: Fifteen puzle game, C89, plain vanillia TTY, MVC, § 22 */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
#define N 4
#define M 4
enum Move{UP,DOWN,LEFT,RIGHT};int hR;int hC;int cc[N][M];const int nS=100;int
update(enum Move m){const int dx[]={0,0,-1,1};const int dy[]={-1,1,0,0};int i=hR
+dy[m];int j=hC+dx[m];if(i>= 0&&i<N&&j>=0&&j<M){cc[hR][hC]=cc[i][j];cc[i][j]=0;
hR=i;hC=j;return 1;}return 0;}void setup(void){int i,j,k;for(i=0;i<N;i++)for(j=0
;j<M;j++)cc[i][j]=i*M+j+1;cc[N-1][M-1]=0;hR=N-1;hC=M-1;k=0;while(k<nS)k+=update(
(enum Move)(rand()%4));}int isEnd(void){int i,j; int k=1;for(i=0;i<N;i++)for(j=0
;j<M;j++)if((k<N*M)&&(cc[i][j]!=k++))return 0;return 1;}void show(){int i,j;
putchar('\n');for(i=0;i<N;i++)for(j=0;j<M;j++){if(cc[i][j])printf(j!=M-1?" %2d "
:" %2d \n",cc[i][j]);else printf(j!=M-1?" %2s ":" %2s \n", "");}putchar('\n');}
void disp(char* s){printf("\n%s\n", s);}enum Move get(void){int c;for(;;){printf
("%s","enter u/d/l/r : ");c=getchar();while(getchar()!='\n');switch(c){case 27:
exit(0);case'd':return UP;case'u':return DOWN;case'r':return LEFT;case'l':return
RIGHT;}}}void pause(void){getchar();}int main(void){srand((unsigned)time(NULL));
do setup();while(isEnd());show();while(!isEnd()){update(get());show();}disp(
"You win"); pause();return 0;}
```

C89, short version, TTY mode

```
/*
 * RosettaCode: Fifteen puzle game, C89, plain vanillia TTY, MVC
 */

#define _CRT_SECURE_NO_WARNINGS /* unlocks printf etc. in MSVC */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

enum Move { MOVE_UP = 0, MOVE_DOWN = 1, MOVE_LEFT = 2, MOVE_RIGHT = 3 };

/* ************************************************************************
 * Model
 */

#define NROWS     4
#define NCOLLUMNS 4
int holeRow;
int holeCollumn;
int cells[NROWS][NCOLLUMNS];
const int nShuffles = 100;

int Game_update(enum Move move){
    const int dx[] = {  0,  0, -1, +1 };
    const int dy[] = { -1, +1,  0,  0 };
    int i = holeRow    + dy[move];
    int j = holeCollumn + dx[move];
    if ( i >= 0 && i < NROWS && j >= 0 && j < NCOLLUMNS ){
        cells[holeRow][holeCollumn] = cells[i][j];
        cells[i][j] = 0; holeRow = i; holeCollumn = j;
        return 1;
    }
    return 0;
}

void Game_setup(void){
    int i,j,k;
    for ( i = 0; i < NROWS; i++ )
        for ( j = 0; j < NCOLLUMNS; j++ )
            cells[i][j] = i * NCOLLUMNS + j + 1;
    cells[NROWS-1][NCOLLUMNS-1] = 0;
    holeRow = NROWS - 1;
    holeCollumn = NCOLLUMNS - 1;
    k = 0;
    while ( k < nShuffles )
        k += Game_update((enum Move)(rand() % 4));
}

int Game_isFinished(void){
    int i,j; int k = 1;
    for ( i = 0; i < NROWS; i++ )
        for ( j = 0; j < NCOLLUMNS; j++ )
            if ( (k < NROWS*NCOLLUMNS) && (cells[i][j] != k++ ) )
                return 0;
```

```
    return 1;
}


/* ************************************************************************
 * View
 */

void View_showBoard(){
    int i,j;
    putchar('\n');
    for ( i = 0; i < NROWS; i++ )
        for ( j = 0; j < NCOLLUMNS; j++ ){
            if ( cells[i][j] )
                printf(j != NCOLLUMNS-1 ? " %2d " : " %2d \n", cells[i][j]);
            else
                printf(j != NCOLLUMNS-1 ? " %2s " : " %2s \n", "");
        }
    putchar('\n');
}

void View_displayMessage(char* text){
    printf("\n%s\n", text);
}


/* ************************************************************************
 * Controller
 */

enum Move Controller_getMove(void){
    int c;
    for(;;){
        printf("%s", "enter u/d/l/r : ");
        c = getchar();
        while( getchar() != '\n' )
            ;
        switch ( c ){
            case 27: exit(EXIT_SUCCESS);
            case 'd' : return MOVE_UP;
            case 'u' : return MOVE_DOWN;
            case 'r' : return MOVE_LEFT;
            case 'l' : return MOVE_RIGHT;
        }
    }
}

void Controller_pause(void){
    getchar();
}

int main(void){

    srand((unsigned)time(NULL));

    do Game_setup(); while ( Game_isFinished() );

    View_showBoard();
    while( !Game_isFinished() ){
        Game_update( Controller_getMove() );
        View_showBoard();
    }

    View_displayMessage("You win");
    Controller_pause();

    return EXIT_SUCCESS;
}
```

Output:

```
  9   1   4   7
  6   5   3   2
 13  10       8
 14  15  11  12
```

```
enter u/d/l/r : u

  9   1   4   7
  6   5   3   2
 13  10  11   8
 14  15      12

enter u/d/l/r : l

  9   1   4   7
  6   5   3   2
 13  10  11   8
 14  15  12

enter u/d/l/r : d

  9   1   4   7
  6   5   3   2
 13  10  11
 14  15  12   8

enter u/d/l/r :
```

C89, long version, TTY/Winapi/ncurses modes

```c
/**
 * RosettaCode: Fifteen puzle game, C89, MS Windows Console API, MVC
 *
 * @version 0.2 (added TTY and ncurses modes)
 */

#define UNDEFINED_WIN32API_CONSOLE
#define UNDEFINED_NCURSES_CONSOLE
#if !defined (TTY_CONSOLE) && !defined(WIN32API_CONSOLE) && !defined(NCURSES_CONSOLE)
#define TTY_CONSOLE
#endif

#define _CRT_SECURE_NO_WARNINGS    /* enable printf etc. */
#define _CRT_NONSTDC_NO_DEPRECATE  /* POSIX functions enabled */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#if defined(NCURSES_CONSOLE)
#include "curses.h"  /* see http://pdcurses.sourceforge.net/ */
#elif defined(WIN32API_CONSOLE)
#define NOGDI                   /* we don't need GDI */
#define WIN32_LEAN_AND_MEAN     /* we don't need OLE etc. */
#include <windows.h>            /* MS Windows stuff */
#include <conio.h>              /* kbhit() and getch() */
#endif

enum Move { MOVE_UP = 0, MOVE_DOWN = 1, MOVE_LEFT = 2, MOVE_RIGHT = 3 };

/* ************************************************************************
 * Model
 */

#define NROWS     4
#define NCOLLUMNS 4
int holeRow;
int holeCollumn;
int cells[NROWS][NCOLLUMNS];
const int nShuffles = 100;

int Game_update(enum Move move){
    const int dx[] = {  0,  0, -1, +1 };
    const int dy[] = { -1, +1,  0,  0 };
    int i = holeRow     + dy[move];
    int j = holeCollumn + dx[move];
    if ( i >= 0 && i < NROWS && j >= 0 && j < NCOLLUMNS ){
        cells[holeRow][holeCollumn] = cells[i][j];
```

```
        cells[i][j] = 0; holeRow = i; holeCollumn = j;
        return 1;
    }
    return 0;
}

void Game_setup(void){
    int i,j,k;
    for ( i = 0; i < NROWS; i++ )
        for ( j = 0; j < NCOLLUMNS; j++ )
            cells[i][j] = i * NCOLLUMNS + j + 1;
    cells[NROWS-1][NCOLLUMNS-1] = 0;
    holeRow = NROWS - 1;
    holeCollumn = NCOLLUMNS - 1;
    k = 0;
    while ( k < nShuffles )
        k += Game_update((enum Move)(rand() % 4));
}

int Game_isFinished(void){
    int i,j; int k = 1;
    for ( i = 0; i < NROWS; i++ )
        for ( j = 0; j < NCOLLUMNS; j++ )
            if ( (k < NROWS*NCOLLUMNS) && (cells[i][j] != k++ ) )
                return 0;
    return 1;
}


/* ***************************************************************************
 * View
 */

int fieldWidth;
#ifdef WIN32API_CONSOLE
HANDLE hConsole;
CONSOLE_SCREEN_BUFFER_INFO csbi;
#endif

void View_setup_base(void)
{
    int i;
    fieldWidth = 0;
    for ( i = NROWS * NCOLLUMNS - 1; i > 0; i /= 10 )
        fieldWidth++;
}

#if defined(TTY_CONSOLE)

void View_setup(void) {
    View_setup_base();
}

void View_showBoard()
{
    int i,j;
    putchar('\n');
    for ( i = 0; i < NROWS; i++ )
        for ( j = 0; j < NCOLLUMNS; j++ ){
            if ( cells[i][j] )
                printf(j != NCOLLUMNS-1 ? " %*d " : " %*d \n", fieldWidth, cells[i][j]);
            else
                printf(j != NCOLLUMNS-1 ? " %*s " : " %*s \n", fieldWidth, "");
        }
    putchar('\n');
}

void View_displayMessage(char* text)
{
    printf("\n%s\n", text);
}

#elif defined(NCURSES_CONSOLE)

void View_setup(void) {
    View_setup_base();
    initscr();
    clear();
```

```
}

void View_showBoard()
{
    int i,j;
    for ( i = 0; i < NROWS; i++ )
        for ( j = 0; j < NCOLLUMNS; j++ ){
            int x = (fieldWidth+1)*j;
            int y = 2*i;
            if ( cells[i][j] ){
                attron(A_REVERSE);
                mvprintw(y,x,"%*d", fieldWidth, cells[i][j]);
            }else{
                attroff(A_REVERSE);
                mvprintw(y,x,"%*s", fieldWidth, " ");
            }
        }
    attrset(A_NORMAL);
}

void View_displayMessage(char* text)
{
    mvprintw(2*NROWS,0, "%s", text);
}

#elif defined(WIN32API_CONSOLE)

void View_setup(void) {
    const COORD coordHome = { 0, 0 };
    CONSOLE_CURSOR_INFO cci;
    DWORD size, nWritten;
    View_setup_base();
    hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    cci.bVisible = FALSE;
    cci.dwSize = 1;
    SetConsoleCursorInfo(hConsole,&cci);
    GetConsoleScreenBufferInfo(hConsole,&(csbi));
    size = csbi.dwSize.X*csbi.dwSize.Y;
    FillConsoleOutputCharacter(hConsole,' ',size,coordHome,&nWritten);
    FillConsoleOutputAttribute(hConsole,csbi.wAttributes,size,coordHome,&nWritten);

}

void View_showBoard()
{
    int i,j;
    char labelString[32];
    WORD attributes;
    DWORD nWritten;
    for ( i = 0; i < NROWS; i++ )
        for ( j = 0; j < NCOLLUMNS; j++ ){
            COORD coord = { ((SHORT)fieldWidth+1)*j, coord.Y = 2*i };
            if ( cells[i][j] ){
                sprintf(labelString,"%*d", fieldWidth, cells[i][j]);
                attributes = BACKGROUND_BLUE | BACKGROUND_GREEN | BACKGROUND_RED;
            }else{
                sprintf(labelString,"%*s", fieldWidth, " ");
                attributes = csbi.wAttributes;
            }
            WriteConsoleOutputCharacter(hConsole,labelString,fieldWidth,coord,&nWritten);
            FillConsoleOutputAttribute (hConsole,attributes,fieldWidth,coord,&nWritten);
        }
}

void View_displayMessage(char* text)
{
    DWORD nWritten;
    COORD coord = { 0, 2 * NROWS };
    WriteConsoleOutputCharacter(hConsole,text,strlen(text),coord,&nWritten);
}

#endif


/* *************************************************************************
 * Controller
 */
```

```
#if defined(TTY_CONSOLE)

void Controller_setup(void){
}

enum Move Controller_getMove(void){
    int c;
    for(;;){
        printf("%s", "enter u/d/l/r : ");
        c = getchar();
        while( getchar() != '\n' )
            ;
        switch ( c ){
            case 27: exit(EXIT_SUCCESS);
            case 'd' : return MOVE_UP;
            case 'u' : return MOVE_DOWN;
            case 'r' : return MOVE_LEFT;
            case 'l' : return MOVE_RIGHT;
        }
    }
}

void Controller_pause(void)
{
    getchar();
}

#elif defined(NCURSES_CONSOLE)

void Controller_setup(void){
    noecho();
    cbreak();
    curs_set(0);
    keypad(stdscr,TRUE);
}

enum Move Controller_getMove(void){
    for(;;){
        switch ( wgetch(stdscr) ){
            case  27: exit(EXIT_SUCCESS);
            case KEY_DOWN  : return MOVE_UP;
            case KEY_UP    : return MOVE_DOWN;
            case KEY_RIGHT : return MOVE_LEFT;
            case KEY_LEFT  : return MOVE_RIGHT;
            case ERR: /* NOP */;
        }
    }
}

void Controller_pause(void){
    while ( wgetch(stdscr) == ERR )
        ;
}


#elif defined(WIN32API_CONSOLE)

void Controller_setup(void){
}

enum Move Controller_getMove(void){
    for(;;){
        switch ( getch() ){
            case  27: exit(EXIT_SUCCESS);
            case   0:
            case 224: switch ( getch() ){
                case 80 : return MOVE_UP;
                case 72 : return MOVE_DOWN;
                case 77 : return MOVE_LEFT;
                case 75 : return MOVE_RIGHT;
            }
        }
    }
}

void Controller_pause(void){
    while(  kbhit() ) getch();
    while( !kbhit() )   ;
```

```
    while(  kbhit() ) getch();
}

#endif


/* *************************************************************************
 * Main function: create model, view and controller. Run main loop.
 */
int main(void) {

    srand((unsigned)time(NULL));

    do Game_setup(); while ( Game_isFinished() );
    View_setup();
    Controller_setup();

    View_showBoard();
    while( !Game_isFinished() ){
        Game_update( Controller_getMove() );
        View_showBoard();
    }

    View_displayMessage("You win");
    Controller_pause();

    return EXIT_SUCCESS;
}
```

C#

Library: System.Windows.Forms
Library: System.Drawing
Works with: C sharp version 6

```
using System;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

public class FifteenPuzzle
{
    const int gridSize = 4; //Standard 15 puzzle is 4x4
    const bool evenSized = gridSize % 2 == 0;
    const int blockCount = gridSize * gridSize;
    const int last = blockCount - 1;
    const int buttonSize = 50;
    const int buttonMargin = 3; //default = 3
    const int formEdge = 9;
    static readonly Random rnd = new Random();
    static readonly Font buttonFont = new Font("Arial", 15.75F, FontStyle.Regular, GraphicsUnit.Point, ((byte)
(0)));
    readonly Button[] buttons = new Button[blockCount];
    readonly int[] grid = new int[blockCount];
    readonly int[] positionOf = new int[blockCount];
    int moves = 0;
    DateTime start;

    public static void Main(string[] args)
    {
        FifteenPuzzle p = new FifteenPuzzle();
        Form f = p.BuildForm();
        Application.Run(f);
    }

    public FifteenPuzzle()
    {
        for (int i = 0; i < blockCount; i++) {
            grid[i] = i;
            positionOf[i] = i;
        }
    }

    Form BuildForm()
```

```
{
    Button startButton = new Button {
        Font = new Font("Arial", 9.75F, FontStyle.Regular, GraphicsUnit.Point, ((byte)(0))),
        Size = new Size(86, 23),
        Location = new Point(formEdge,
            (buttonSize + buttonMargin * 2) * gridSize + buttonMargin + formEdge),
        Text = "New Game",
        UseVisualStyleBackColor = true
    };
    startButton.Click += (sender, e) => Shuffle();

    int size = buttonSize * gridSize + buttonMargin * gridSize * 2 + formEdge * 2;
    Form form = new Form {
        Text = "Fifteen",
        ClientSize = new Size(width: size, height: size + buttonMargin * 2 + startButton.Height)
    };
    form.SuspendLayout();
    for (int index = 0; index < blockCount; index++) {
        Button button = new Button {
            Font = buttonFont,
            Size = new Size(buttonSize, buttonSize),
            //Margin = new Padding(buttonMargin),
            Text = (index + 1).ToString(),
            UseVisualStyleBackColor = true
        };
        SetLocation(button, index);
        form.Controls.Add(button);
        buttons[index] = button;
        int i = index;
        button.Click += (sender, e) => ButtonClick(i);
    }
    form.Controls.Add(startButton);
    form.ResumeLayout();
    return form;
}

void ButtonClick(int i)
{
    if (buttons[last].Visible) return;
    int target = positionOf[i];
    if (positionOf[i] / gridSize == positionOf[last] / gridSize) {
        while (positionOf[last] < target) {
            Swap(last, grid[positionOf[last] + 1]);
            moves++;
        }
        while (positionOf[last] > target) {
            Swap(last, grid[positionOf[last] - 1]);
            moves++;
        }
    } else if (positionOf[i] % gridSize == positionOf[last] % gridSize) {
        while (positionOf[last] < target) {
            Swap(last, grid[positionOf[last] + gridSize]);
            moves++;
        }
        while (positionOf[last] > target) {
            Swap(last, grid[positionOf[last] - gridSize]);
            moves++;
        }
    }
    if (Solved()) {
        TimeSpan elapsed = DateTime.Now - start;
        elapsed = TimeSpan.FromSeconds(Math.Round(elapsed.TotalSeconds, 0));
        buttons[last].Visible = true;
        MessageBox.Show($"Solved in {moves} moves. Time: {elapsed}");
    }
}

bool Solved() => Enumerable.Range(0, blockCount - 1).All(i => positionOf[i] == i);

static void SetLocation(Button button, int index)
{
    int row = index / gridSize, column = index % gridSize;
    button.Location = new Point(
        (buttonSize + buttonMargin * 2) * column + buttonMargin + formEdge,
        (buttonSize + buttonMargin * 2) * row + buttonMargin + formEdge);
}

void Shuffle()
```

```
    {
        for (int i = 0; i < blockCount; i++) {
            int r = rnd.Next(i, blockCount);
            int g = grid[r];
            grid[r] = grid[i];
            grid[i] = g;
        }
        for (int i = 0; i < blockCount; i++) {
            positionOf[grid[i]] = i;
            SetLocation(buttons[grid[i]], i);
        }
        if (!Solvable()) Swap(0, 1); //Swap any 2 blocks

        buttons[last].Visible = false;
        moves = 0;
        start = DateTime.Now;
    }

    bool Solvable()
    {
        bool parity = true;
        for (int i = 0; i < blockCount - 2; i++) {
            for (int j = i + 1; j < blockCount - 1; j++) {
                if (positionOf[j] < positionOf[i]) parity = !parity;
            }
        }
        if (evenSized && positionOf[last] / gridSize % 2 == 0) parity = !parity;
        return parity;
    }

    void Swap(int a, int b)
    {
        Point location = buttons[a].Location;
        buttons[a].Location = buttons[b].Location;
        buttons[b].Location = location;

        int p = positionOf[a];
        positionOf[a] = positionOf[b];
        positionOf[b] = p;

        grid[positionOf[a]] = a;
        grid[positionOf[b]] = b;
    }
}
```

C++

```
#include <time.h>
#include <stdlib.h>
#include <vector>
#include <string>
#include <iostream>
class p15 {
public :
    void play() {
        bool p = true;
        std::string a;
        while( p ) {
            createBrd();
            while( !isDone() ) { drawBrd();getMove(); }
            drawBrd();
            std::cout << "\n\nCongratulations!\nPlay again (Y/N)?";
            std::cin >> a; if( a != "Y" && a != "y" ) break;
        }
    }
private:
    void createBrd() {
        int i = 1; std::vector<int> v;
        for( ; i < 16; i++ ) { brd[i - 1] = i; }
        brd[15] = 0; x = y = 3;
        for( i = 0; i < 1000; i++ ) {
            getCandidates( v );
            move( v[rand() % v.size()] );
            v.clear();
```

```
        }
    }
    void move( int d ) {
        int t = x + y * 4;
        switch( d ) {
            case 1: y--; break;
            case 2: x++; break;
            case 4: y++; break;
            case 8: x--;
        }
        brd[t] = brd[x + y * 4];
        brd[x + y * 4] = 0;
    }
    void getCandidates( std::vector<int>& v ) {
        if( x < 3 ) v.push_back( 2 ); if( x > 0 ) v.push_back( 8 );
        if( y < 3 ) v.push_back( 4 ); if( y > 0 ) v.push_back( 1 );
    }
    void drawBrd() {
        int r; std::cout << "\n\n";
        for( int y = 0; y < 4; y++ ) {
            std::cout << "+----+----+----+----+\n";
            for( int x = 0; x < 4; x++ ) {
                r = brd[x + y * 4];
                std::cout << "| ";
                if( r < 10 ) std::cout << " ";
                if( !r ) std::cout << "  ";
                else std::cout << r << " ";
            }
            std::cout << "|\n";
        }
        std::cout << "+----+----+----+----+\n";
    }
    void getMove() {
        std::vector<int> v; getCandidates( v );
        std::vector<int> p; getTiles( p, v ); unsigned int i;
        while( true ) {
            std::cout << "\nPossible moves: ";
            for( i = 0; i < p.size(); i++ ) std::cout << p[i] << " ";
            int z; std::cin >> z;
            for( i = 0; i < p.size(); i++ )
                if( z == p[i] ) { move( v[i] ); return; }
        }
    }
    void getTiles( std::vector<int>& p, std::vector<int>& v ) {
        for( unsigned int t = 0; t < v.size(); t++ ) {
            int xx = x, yy = y;
            switch( v[t] ) {
                case 1: yy--; break;
                case 2: xx++; break;
                case 4: yy++; break;
                case 8: xx--;
            }
            p.push_back( brd[xx + yy * 4] );
        }
    }
    bool isDone() {
        for( int i = 0; i < 15; i++ ) {
            if( brd[i] != i + 1 ) return false;
        }
        return true;
    }
    int brd[16], x, y;
};
int main( int argc, char* argv[] ) {
    srand( ( unsigned )time( 0 ) );
    p15 p; p.play(); return 0;
}
```

```
+----+----+----+----+
| 11 |  5 | 12 |  3 |
+----+----+----+----+
| 10 |  7 |  6 |  4 |
+----+----+----+----+
| 13 |    |  2 |  1 |
+----+----+----+----+
| 15 | 14 |  8 |  9 |
```

```
+----+----+----+----+

Possible moves: 2 13 14 7
```

COBOL

Tested with GnuCOBOL

```
         >>SOURCE FORMAT FREE
*> This code is dedicated to the public domain
*> This is GNUCOBOL 2.0
identification division.
program-id. fifteen.
environment division.
configuration section.
repository. function all intrinsic.
data division.
working-storage section.

01  r pic 9.
01  r-empty pic 9.
01  r-to pic 9.
01  r-from pic 9.

01  c pic 9.
01  c-empty pic 9.
01  c-to pic 9.
01  c-from pic 9.

01  display-table.
    03  display-row occurs 4.
        05  display-cell occurs 4 pic 99.

01  tile-number pic 99.
01  tile-flags pic x(16).

01  display-move value spaces.
    03  tile-id pic 99.

01  row-separator pic x(21) value all '.'.
01  column-separator pic x(3) value ' . '.

01  inversions pic 99.
01  current-tile pic 99.

01  winning-display pic x(32) value
        '01020304'
    &   '05060708'
    &   '09101112'
    &   '13141500'.

procedure division.
start-fifteen.
    display 'start fifteen puzzle'
    display '    enter a two-digit tile number and press <enter> to move'
    display '    press <enter> only to exit'

    *> tables with an odd number of inversions are not solvable
    perform initialize-table with test after until inversions = 0
    perform show-table
    accept display-move
    perform until display-move = spaces
        perform move-tile
        perform show-table
        move spaces to display-move
        accept display-move
    end-perform
    stop run
    .
initialize-table.
    compute tile-number = random(seconds-past-midnight) *> seed only
    move spaces to tile-flags
    move 0 to current-tile inversions
```

```
    perform varying r from 1 by 1 until r > 4
    after c from 1 by 1 until c > 4
        perform with test after
        until tile-flags(tile-number + 1:1) = space
            compute tile-number = random() * 100
            compute tile-number = mod(tile-number, 16)
        end-perform
        move 'x' to tile-flags(tile-number + 1:1)
        if tile-number > 0 and < current-tile
            add 1 to inversions
        end-if
        move tile-number to display-cell(r,c) current-tile
    end-perform
    compute inversions = mod(inversions,2)
    .
show-table.
    if display-table = winning-display
        display 'winning'
    end-if
    display space row-separator
    perform varying r from 1 by 1 until r > 4
        perform varying c from 1 by 1 until c > 4
            display column-separator with no advancing
            if display-cell(r,c) = 00
                display '  ' with no advancing
                move r to r-empty
                move c to c-empty
            else
                display display-cell(r,c) with no advancing
            end-if
        end-perform
        display column-separator
    end-perform
    display space row-separator
    .
move-tile.
    if not (tile-id numeric and tile-id >= 01 and <= 15)
        display 'invalid tile number'
        exit paragraph
    end-if

    *> find the entered tile-id row and column (r,c)
    perform varying r from 1 by 1 until r > 4
    after c from 1 by 1 until c > 4
        if display-cell(r,c) = tile-id
            exit perform
        end-if
    end-perform

    *> show-table filled (r-empty,c-empty)
    evaluate true
    when r = r-empty
        if c-empty < c
            *> shift left
            perform varying c-to from c-empty by 1 until c-to > c
                compute c-from = c-to + 1
                move display-cell(r-empty,c-from) to display-cell(r-empty,c-to)
            end-perform
        else
           *> shift right
           perform varying c-to from c-empty by -1 until c-to < c
               compute c-from = c-to - 1
               move display-cell(r-empty,c-from) to display-cell(r-empty,c-to)
           end-perform
       end-if
       move 00 to display-cell(r,c)
    when c = c-empty
        if r-empty < r
            *>shift up
            perform varying r-to from r-empty by 1 until r-to > r
                compute r-from = r-to + 1
                move display-cell(r-from,c-empty) to display-cell(r-to,c-empty)
            end-perform
        else
            *> shift down
            perform varying r-to from r-empty by -1 until r-to < r
                compute r-from = r-to - 1
                move display-cell(r-from,c-empty) to display-cell(r-to,c-empty)
```

```
            end-perform
        end-if
        move 00 to display-cell(r,c)
    when other
         display 'invalid move'
    end-evaluate
    .
end program fifteen.
```

Output:

```
prompt$ cobc -xj fifteen.cbl
start fifteen puzzle
    enter a two-digit tile number and press <enter> to move
    press <enter> only to exit
 ....................
 . 05 . 14 . 08 . 12 .
 . 01 . 10 . 03 . 09 .
 . 02 . 15 . 13 . 11 .
 . 06 .    . 07 . 04 .
 ....................
```

Common Lisp

Credit to this post for help with the inversions-counting function: [1] (http://www.lispforum.com/viewtopic.php?f=2&t=3422)

Run it (after loading the file) with

```
|15|::main
```

.

```
(defpackage :15
  (:use :common-lisp))
(in-package :15)

(defvar +side+ 4)
(defvar +max+ (1- (* +side+ +side+))) ; 15

(defun make-board ()
  (make-array (list +side+ +side+)
              :initial-contents
              (loop :for i :below +side+ :collecting
                 (loop :for j :below +side+ :collecting
                    (mod (1+ (+ j (* i +side+))) (1+ +max+))))))
(defvar *board* (make-board))

(defun shuffle-board (board)
  (loop for i from (array-total-size board) downto 2
     do (rotatef (row-major-aref board (random i))
                 (row-major-aref board (1- i))))
  board)

(defun pb (stream object &rest args)
  (declare (ignorable args))
  (loop for i below (car (array-dimensions object)) do
       (loop for j below (cadr (array-dimensions object)) do
            (let ((cell (aref object i j)))
              (format stream "(~[  ~:;~:*~2d~])" cell)))
       (format stream "~%")))

(defun sortedp (board)
  (declare (ignorable board))
  (loop for i upto +max+
     when (eq (row-major-aref board i) (mod (1+ i) 16)) do
       (return-from sortedp nil))
  t)
```

```
(defun inversions (lst)
  (if (or (null lst) (null (cdr lst)))
      0
      (let* ((half (ceiling (/ (length lst) 2)))
             (left-list (subseq lst 0 half))
             (right-list (subseq lst half)))
        (+ (loop for a in left-list
              summing (loop for b in right-list
                        counting (not (< a b))))
           (inversions left-list)
           (inversions right-list)))))

(defun solvablep (board)
  (let ((inv (inversions (loop for i upto +max+ collecting
                               (row-major-aref board i))))
        (row (- +side+ (first (board-position board 0)))))
    (or (and (oddp +side+)
             (evenp inv))
        (and (evenp +side+)
             (evenp row)
             (oddp inv))
        (and (evenp +side+)
             (oddp row)
             (evenp inv)))))

(defun board-position (board dig)
  (loop for i below (car (array-dimensions board)) do
       (loop for j below (cadr (array-dimensions board))
          when (eq dig (aref board i j)) do
          (return-from board-position (list i j)))))

(defun in-bounds (y x)
  (and (< -1 y +side+)
       (< -1 x +side+)))

(defun get-adjacents (board pos)
  (let ((adjacents ()) (y (first pos)) (x (second pos)))
    (if (in-bounds y (1+ x))
        (push (aref board y (1+ x)) adjacents))
    (if (in-bounds (1+ y) x)
        (push (aref board (1+ y) x) adjacents))
    (if (in-bounds y (1- x))
        (push (aref board y (1- x)) adjacents))
    (if (in-bounds (1- y) x)
        (push (aref board (1- y) x) adjacents))
    adjacents))

(defun main (&rest argv)
  (declare (ignorable argv))
  (setf *random-state* (make-random-state t))
  (loop until (solvablep *board*) do
       (shuffle-board *board*))
  (loop until (sortedp *board*) do
       (format t "~/15:pb/~%" *board*)
       (format t "Which number do you want to swap the blank with?~%> ")
       (let* ((in (read))
              (zpos (board-position *board* 0))
              (pos (board-position *board* in))
              (adj (get-adjacents *board* zpos)))
         (if (find in adj)
             (rotatef (aref *board* (first pos) (second pos))
                      (aref *board* (first zpos) (second zpos))))))
  (format t "You win!~%"))
```

EasyLang

Run it (https://easylang.online/apps/15-puzzle.html)

```
background 432
textsize 13
len f[] 16
func draw . .
```

```
  clear
  for i = 1 to 16
    h = f[i]
    if h < 16
      x = (i - 1) mod 4 * 24 + 3
      y = (i - 1) div 4 * 24 + 3
      color 210
      move x y
      rect 22 22
      move x + 4 y + 6
      if h < 10
        move x + 6 y + 6
      .
      color 885
      text h
    .
  .
.
global done .
func init . .
  done = 0
  for i = 1 to 16
    f[i] = i
  .
  # shuffle
  for i = 15 downto 2
    r = random i
    swap f[r] f[i]
  .
  # make it solvable
  inv = 0
  for i = 1 to 15
    for j = 1 to i - 1
      if f[j] > f[i]
        inv += 1
      .
    .
  .
  if inv mod 2 <> 0
    swap f[1] f[2]
  .
  textsize 12
  call draw
.
func move_tile . .
  c = mouse_x div 25
  r = mouse_y div 25
  i = r * 4 + c + 1
  if c > 0 and f[i - 1] = 16
    swap f[i] f[i - 1]
  elif r > 0 and f[i - 4] = 16
    swap f[i] f[i - 4]
  elif r < 3 and f[i + 4] = 16
    swap f[i] f[i + 4]
  elif c < 3 and f[i + 1] = 16
    swap f[i] f[i + 1]
  .
  call draw
  done = 1
  for i = 1 to 15
    if f[i] > f[i + 1]
      done = 0
    .
  .
  if done = 1
    clear
    move 10 30
    text "Well done!"
  .
.
on mouse_down
  if done = 1
    call init
  else
    call move_tile
  .
```

```
.
call init
```

F#

```
// 15 Puzzle Game. Nigel Galloway: August 9th., 2020
let Nr,Nc,RA,rnd=[|3;0;0;0;0;1;1;1;1;2;2;2;2;3;3;3|],[|3;0;1;2;3;0;1;2;3;0;1;2;3;0;1;2|],[|for n in [1..16]->n%16|],System.Random()
let rec fN g Σ=function h::t->fN g (Σ+List.sumBy(fun n->if h>n then 1 else 0)t) t|_->(Σ-g/4)%2=1
let rec fI g=match if System.Console.IsInputRedirected then char(System.Console.Read()) else System.Console.ReadKey(true).KeyChar with
             n when Seq.contains n g->printf "%c" n; (match n with 'l'-> -1|'r'->1|'d'->4|_-> -4)|_->System.Console.Beep(); fI g
let rec fG n Σ=function 0->(List.findIndex((=)0)Σ,Σ)|g->let i=List.item(rnd.Next(g)) n in fG(List.except [i] n)(i::Σ)(g-1)
let rec fE()=let n,g=fG [0..15] [] 16 in if fN n 0 (List.except [0] g) then (n,Array.ofList g) else fE()
let rec fL(n,g) Σ=let fa=printfn "";Array.chunkBySize 4 g|>Array.iter(fun n->Array.iter(printf "%3d")n;printfn "")
                  match g=RA with true->printfn "Solved in %d moves" Σ; fa; 0
                                  |_->let vM=match n/4,n%4 with (0,0)->"rd"|(0,3)->"ld"|(0,_)->"lrd"|(3,0)->"ru"|(3,3)->"lu"|(3,_)->"lru"|(_,0)->"rud"|(_,3)->"lud"|_->"lrud"
                                      fa; printf "Move Number: %2d; Manhatten Distance: %2d; Choose move from %4s: " Σ
                                          ([0..15]|>List.sumBy(fun n->match g.[n] with 0->0 |i->(abs(n/4-Nr.[i]))+(abs(n%4-Nc.[i])))) vM
                                      let v=fI vM in g.[n]<-g.[n+v];g.[n+v]<-0;fL(n+v,g)(Σ+1)
[<EntryPoint>]
let main n = fL(match n with [|n|]->let g=[|let g=uint64 n in for n in 60..-4..0->int((g>>>n)&&&15UL)|] in (Array.findIndex((=)0)g,g) |_->fE()) 0
```

3 uses:

```
15 game with no parameters will generate a random game which may be solved.
A solution in the form suggested in 15_puzzle_solver may be piped into 15game.
"dddrurdruuulllddrulddrrruuullddruullddrrurulldrruulldldddrurullddrrruullulddrdrr" | .\15game 0x0c9dfbae37254861
produces the output here: 15_puzzle_solver/extra_credit/solution.
Optionally a start position may be supplied. 15game 0x123450689a7bdefc may be solved:
```

Output:

```
  1  2  3  4
  5  0  6  8
  9 10  7 11
 13 14 15 12
Move Number:  0; Manhatten Distance:  4; Choose move from lrud: r
  1  2  3  4
  5  6  0  8
  9 10  7 11
 13 14 15 12
Move Number:  1; Manhatten Distance:  3; Choose move from lrud: d
  1  2  3  4
  5  6  7  8
  9 10  0 11
 13 14 15 12
Move Number:  2; Manhatten Distance:  2; Choose move from lrud: r
  1  2  3  4
  5  6  7  8
  9 10 11  0
 13 14 15 12
Move Number:  3; Manhatten Distance:  1; Choose move from  lud: d
  1  2  3  4
  5  6  7  8
  9 10 11 12
```

```
 13 14 15  0
Solved in 4 moves
```

Factor

```
USING: accessors combinators combinators.extras
combinators.short-circuit grouping io kernel literals math
math.matrices math.order math.parser math.vectors prettyprint qw
random sequences sequences.extras ;
IN: rosetta-code.15-puzzle-game

<<

TUPLE: board matrix zero ;

: <board> ( -- board )
    16 <iota> 1 rotate 4 group { 3 3 } board boa ;

>>

CONSTANT: winning $[ <board> matrix>> ]

: input>dir ( str -- pair )
    {
        { "u" [ { 1 0 } ] }
        { "d" [ { -1 0 } ] }
        { "l" [ { 0 1 } ] }
        { "r" [ { 0 -1 } ] }
    } case ;

: get-index ( loc matrix -- elt ) [ first2 swap ] dip nth nth ;

: mexchange ( loc1 loc2 matrix -- )
    tuck [ [ [ get-index ] keepd ] 2bi@ ] keep [ spin ] 2dip
    [ set-index ] keep set-index ;

: vclamp+ ( seq1 seq2 -- seq ) v+ { 0 0 } { 3 3 } vclamp ;

: slide-piece ( board str -- )
    over zero>> [ vclamp+ ] keep rot matrix>> mexchange ;

: move-zero ( board str -- )
    [ vclamp+ ] curry change-zero drop ;

: move ( board str -- )
    input>dir [ slide-piece ] [ move-zero ] 2bi ;

: rand-move ( board -- ) qw{ u d l r } random move ;

: shuffle-board ( board n -- board' ) [ dup rand-move ] times ;

: .board ( board -- ) matrix>> simple-table. ;

: get-input ( -- str )
    "Your move? (u/d/l/r/q) " write flush readln dup
    qw{ u d l r q } member? [ drop get-input ] unless ;

: won? ( board -- ? ) matrix>> winning = ;

DEFER: game
: process-input ( board -- board' )
    get-input dup "q" = [ drop ] [ game ] if ;

: check-win ( board -- board' )
    dup won? [ "You won!" print ] [ process-input ] if ;

: game ( board str -- board' )
    [ move ] keepd dup .board check-win ;

: valid-difficulty? ( obj -- ? )
    { [ fixnum? ] [ 3 100 between? ] } 1&& ;

: choose-difficulty ( -- n )
    "How many shuffles? (3-100) " write flush readln
```

```
    string>number dup valid-difficulty?
    [ drop choose-difficulty ] unless ;

: main ( -- )
    <board> choose-difficulty shuffle-board dup .board check-win
    drop ;

MAIN: main
```

Output:

```
How many shuffles? (3-100) 5
1  2  3  4
5  6  7  8
9  0  10 12
13 14 11 15
Your move? (u/d/l/r/q) apple
Your move? (u/d/l/r/q) l
1  2  3  4
5  6  7  8
9  10 0  12
13 14 11 15
Your move? (u/d/l/r/q) u
1  2  3  4
5  6  7  8
9  10 11 12
13 14 0  15
Your move? (u/d/l/r/q) l
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 0
You won!
```

Forth

Works with: gforth version 0.7.2

The code tested with gforth 0.7.2. It required a 64-bit system.

See 15_puzzle_solver#Forth for a solver based on the same code.

```
#! /usr/bin/gforth

cell 8 <> [if] s" 64-bit system required" exception throw [then]

\ In the stack comments below,
\ "h" stands for the hole position (0..15),
\ "s" for a 64-bit integer representing a board state,
\ "t" a tile value (0..15, 0 is the hole),
\ "b" for a bit offset of a position within a state,
\ "m" for a masked value (4 bits selected out of a 64-bit state),
\ "w" for a weight of a current path,
\ "d" for a direction constant (0..3)

\ Utility
: 3dup   2 pick 2 pick 2 pick ;
: 4dup   2over 2over ;
: shift   dup 0 > if lshift else negate rshift then ;

hex 123456789abcdef0 decimal constant solution
: row   2 rshift ;   : col   3 and ;

: up-valid?    ( h -- f ) row 0 > ;
: down-valid?  ( h -- f ) row 3 < ;
: left-valid?  ( h -- f ) col 0 > ;
: right-valid? ( h -- f ) col 3 < ;

\ To iterate over all possible directions, put direction-related functions into arrays:
: ith ( u addr -- w ) swap cells + @ ;
```

```
create valid? ' up-valid? , ' left-valid? , ' right-valid? , ' down-valid? , does> ith execute ;
create step -4 , -1 , 1 , 4 , does> ith ;

\ Advance from a single state to another:
: bits ( h -- b ) 15 swap - 4 * ;
: tile ( s b -- t ) rshift 15 and ;
: new-state ( s h d -- s' ) step dup >r + bits 2dup tile ( s b t ) swap lshift tuck - swap r> 4 * shift + ;

: hole? ( s u -- f ) bits tile 0= ;
: hole ( s -- h ) 16 0 do dup i hole? if drop i unloop exit then loop drop ;

0 constant up 1 constant left 2 constant right 3 constant down

\ Print a board:
: .hole   space space space ;
: .tile ( u -- ) ?dup-0=-if .hole else dup 10 < if space then . then ;
: .board ( s -- ) 4 0 do cr 4 0 do dup j 4 * i + bits tile .tile loop loop drop ;
: .help   cr ." ijkl move, q quit" ;

\ Pseudorandom number generator:
create (rnd)   utime drop ,
: rnd   (rnd) @ dup 13 lshift xor dup 17 rshift xor dup dup 5 lshift xor (rnd) ! ;

: move ( s u -- s' ) >r dup hole r> new-state ;
: ?move ( s u -- s' ) >r dup hole r@ valid? if r> move else rdrop then ;
: shuffle ( s u -- s' ) 0 do rnd 3 and ?move loop ;

: win   cr ." you won!" bye ;
: turn ( s -- )
    page dup .board .help
    key case
        [char] q of bye endof
        [char] i of down ?move endof
        [char] j of right ?move endof
        [char] k of up ?move endof
        [char] l of left ?move endof
    endcase ;

: play  begin dup solution <> while turn repeat win ;

solution 1000 shuffle play
```

Fortran

The initial version had me so enamoured by the notion of consecutive cells for the solution having the number of their index as their value (as in CELL(0) = 0 (the blank square), CELL(1) = 1, ... CELL(15) = 15) and the prospect of the check for this being simple, that I failed to perceive that the nice big diagram of the board shown at the head of the article in fact clearly shows the solution state having the blank cell at the *end*, not the start. Once again it is demonstrated that what you see is ... influenced ... by what you would like to see. After that diversion, the cells shall now be numbered one to sixteen, not zero to fifteen, and so there is no need for the ability introduced by F90 whereby arrays can have a lower bound other than one.

The plan is to use parameters for the board size, which need not be square. As often with Fortran, messing with arrays is the key, though not without opportunities for confusion. Because Fortran stores arrays in column-major order, the arrays are accessed as BOARD(column,row) even though the arrangement is treated as rows down the page and columns across as is usual. By this means, consecutive elements in storage of array BOARD(c,r) are such that the same storage accessed via array BORED(i) thanks to `EQUIVALENCE(BOARD,BORED)` indexes them as consecutive elements, and so the test that the values are in consecutive order becomes delightfully simple, though alas there is no equivalent of the *iota* function of APL whereby the test could be `ALL(BORED(1:N - 1) .EQ. IOTA(N - 1))`

Column-major ordering also applies to array WAY, which lists the offsets needed to locate squares deemed adjacent to a given location, such as that of the blank square, located by LOCI = LOCZ + WAY(i). Adjacent LOCI are checked for being in range, and if so, added to the list in array LOCM

with the moveable piece identified in array MOVE.

It transpires that the F90 compiler will not allow a PARAMETER statement to define values for items appearing in an EQUIVALENCE statement; so much for an attempt to do so in a systematic manner employing related names.

The game plan is to start with an ordered array so that each cell definitely has a unique code, then jumble them via "random" swaps. Possible arrangements turn out to have either odd or even parity based on the number of out-of-sequence squares, and as the allowed transformations do not change the parity and the solution state has even parity, odd parity starting states should not be presented except by those following Franz Kafka. The calculation is simplified by always having the blank square in the last position, thus in the last row. Once an even-parity starting state is floundered upon, the blank square is re-positioned using allowable moves so that the parity is not altered thereby. Then the game begins: single-square moves only are considered, though in practice groups of squares could be moved horizontally or vertically rather than one-step-at-a-time - a possible extension.

The source style uses F90 for its array arithmetic abilities, especially the functions ALL, ANY and COUNT. A statement

```
LOCZ = MINLOC(BOARD)    !Find the zero. 0 = BOARD(LOCZ(1),LOCZ(2)) == BOARD(ZC,ZR)
```

could be used but is unnecessary thanks to tricks with EQUIVALENCE. For earlier Fortran, various explicit DO-loops would have to be used. This would at least make clear whether or not the equivalents of ANY and ALL terminated on the first failure or doggedly scanned the entire array no matter what.

```
      SUBROUTINE SWAP(I,J)  !Alas, furrytran does not provide this.
       INTEGER I,J,T        !So, we're stuck with supplying the obvious.
        T = I           !And, only for one type at a go.
        I = J           !One could define a MODULE containing a collection
        J = T           !And thence a "generic" routine,
      END SUBROUTINE SWAP   !But this will do for now.

      SUBROUTINE SHOW(NR,NC,BOARD)  !The layout won't work for NC > 99...
       INTEGER NR,NC        !Number of rows and columns.
       INTEGER BOARD(NC,NR) !The board is stored transposed!
       INTEGER I        !A stepper.
       COMMON/IODEV/ MSG    !I talk to the trees...
        WRITE (MSG,1) (I,I = 1,NC)  !Prepare a heading.
    1   FORMAT ("Row|",9("__",I1,:),90("_",I2,:))   !This should suffice.
        DO I = 1,NR     !Chug down the rows.
          WRITE (MSG,2) I,BOARD(1:NC,I) !The columns of the row. Usage is BOARD(column,row).
    2     FORMAT (I3,"|",99I3)  !Could use parameters, but enough.
        END DO          !On to the next row.
      END SUBROUTINE SHOW   !That was nice.

      PROGRAM PUZZLE
      INTEGER LOCN(2),NR,NC,N   !Describes the shape of the board.
      INTEGER LOCZ(2),ZC,ZR !Fingers the location of the "blank" square.
      INTEGER LOCI(2),IC,IR !Fingers a location.
Can't EQUIVALENCE (LOCN(1),NC),(LOCN(2),NR) !This usage and a PARAMETER statement is too scary.
      EQUIVALENCE (LOCZ(1),ZC),(LOCZ(2),ZR) !Annotate my (column,row) usage.
      EQUIVALENCE (LOCI(1),IC),(LOCI(2),IR) !Rather than the displayed (row,column) style.
      PARAMETER (NR = 4, NC = 4, N = NR*NC) !Determine the shape of the board.
      INTEGER BOARD(NC,NR)      !Thus transpose furrytran's column-major usage. Beware!!!
      INTEGER BORED(N)          !This allows for consecutive comparisons.
      EQUIVALENCE (BOARD,BORED)     !Because the arrays are in the same place.
      INTEGER WAYS          !Now define adjacency.
      PARAMETER (WAYS = 4)      !Just orthogonal neghbours.
      INTEGER WAY(2,WAYS)       !Now list the allowed adjacencies.
      PARAMETER (WAY = (/1,0, 0,1, -1,0, 0,-1/))    !W(1,1), W(2,1), W(1,2), W(2,2), W(1,3), ...
      INTEGER M,MOVE(WAYS),LOCM(2,WAYS) !Move possibilities.
      INTEGER SPACE         !Document the empty square's code number.
      PARAMETER (SPACE = 0)     !Zero will do.
      INTEGER I,IT,PARITY,TRY       !Odds and ends.
      REAL VALUE            !Humph. Yet another interface to a "random" number generator.
```

```
      COMMON/IODEV/ MSG,KBD !Pass the word.

      KBD = 5   !Standard input. (Keyboard -> Display screen)
      MSG = 6   !Standard output. (Display screen)
      WRITE (MSG,1) NR,NC   !Announce.
    1 FORMAT ("To play 'slide-square' with ",I0," rows and ",
     1 I0," columns.",/,"The game is to slide a square into the space",/
     2 "(thus leaving a space behind) until you attain"/
     3 "the nice orderly layout as follows:",/)
Concoct a board layout.
   10 FOR ALL (I = 1:N - 1) BORED(I) = I    !Prepare the board. Definitely unique values.
      BORED(N) = SPACE          !The empty square is not at the start! Oh well.
      CALL SHOW(NR,NC,BOARD)    !Reveal the nice layout.
   11 DO I = 1,N - 1        !Now shuffle the squares a bit.
        CALL RANDOM(VALUE)      !0 <= VALUE < 1.
        IT = VALUE*(N - 1) + 1      !1 <= IT < N. Don't round up!
        IF (I.NE.IT) CALL SWAP(BORED(I),BORED(IT))  !Whee!
      END DO            !On to the next victim, leaving the last cell alone.
Calculate the parity, knowing the space is at the end. The target state has even parity, with zero inversions.
      PARITY = 0    !There are two classes of arrangements, that can't mix.
      DO I = 1,N - 2    !Skipping the blank cell still at BORED(N).
        PARITY = PARITY + COUNT(BORED(I) > BORED(I + 1:N - 1))  !For each square,
      END DO        !Count the inversions following.
      IF (MOD(PARITY,2).NE.0) GO TO 11  !No transition can change the parity, so, try for another arrangement.
Choose a new position for the space. Using approved moves will not change the parity.
      CALL RANDOM(VALUE)        !0 <= VALUE < 1.
      ZC = VALUE*(NC - 2) + 1       !1 <= ZC < NC: Choose a random column other than the last.
      BOARD(ZC + 1:NC,NR) = BOARD(ZC:NC - 1,NR) !Shift the end of the last row back one place.
      BOARD(ZC,NR) = SPACE          !Put the space in the hole.
      CALL RANDOM(VALUE)            !So the parity doesn't change.
      ZR = VALUE*(NR - 2) + 1       !1 <= ZR < NR: Choose a random row, other than the last.
      BOARD(ZC,ZR + 1:NR) = BOARD(ZC,ZR:NR - 1) !Shift the end of column ZC up one.
      BOARD(ZC,ZR) = SPACE          !Revive the space again.
Cast forth the starting position.
      WRITE (MSG,12)        !Announce the result.
   12 FORMAT (/,"But, your board looks like this...")   !Alas. Almost certainly not in order.
      CALL SHOW(NR,NC,BOARD)    !Just so.
      TRY = 0       !No moves made yet.

Consider possible moves.
   20 TRY = TRY + 1 !Here we go again.
      M = 0     !No moveable pieces are known.
      DO I = 1,WAYS !So scan the possible ways away from LOCZ.
        LOCI = LOCZ + WAY(1:2,I)    !Finger an adjacent location, via the adjacency offsets in array WAY.
        IF (ALL(LOCI > 0) .AND. ALL(LOCI <= (/NC,NR/))) THEN    !Within bounds?
          M = M + 1         !Yes. We have a candidate.
          MOVE(M) = BOARD(IC,IR)    !Record the piece's name.
          LOCM(:,M) = LOCI      !And, remember where it is...
        END IF          !So much for that location.
      END DO        !Try another offset.
   21 WRITE (MSG,22,ADVANCE="no") MOVE(1:M) !Two-stage output.
   22 FORMAT ("Moveable pieces: ",<WAYS>(I0:","))   !Since M is not necessarily WAYS, a trailing $ may not be
reached..
      WRITE (MSG,23)        !Now for the question. Always at least two moveable squares.
   23 FORMAT(". Choose one: ",$)    !Continue the line, presuming screen and keyboard->screen.
      READ (KBD,*) IT       !Now request the answer. Rather doggedly: blank lines won't do.
      DO I = M,1,-1     !There are at least two possible moves.
        IF (MOVE(I) .EQ. IT) EXIT   !Perhaps this piece was selected.
      END DO            !The INDEX function is alas, only for CHARACTER variables. Grr.
      IF (I .LE. 0) THEN    !I'm suspicious.
        WRITE (MSG,*) "Huh? That is not a moveable piece!"  !Justified!
        IF (IT.GT.0) GO TO 21       !Try again.
        STOP "Oh well."         !Or quit, on negative vibrations.
      END IF            !So much for selecting a piece.
Complete the selected move.
   30 BOARD(ZC,ZR) = IT     !Place the named piece where the space was.
      LOCZ = LOCM(:,I)      !The space is now where that piece came from.
      BOARD(ZC,ZR) = SPACE      !And now holds a space.
c      write (6,*)
c     1 "disorder=",COUNT(BORED(1:N - 2) + 1 .NE. BORED(2:N - 1))
      IF (TRY.LE.6) WRITE (MSG,31)  !Set off with a nice heading.
   31 FORMAT (/"The new layout...") !Just for clarity.
      CALL SHOW(NR,NC,BOARD)        !Perhaps it will be good.
Check for success.
      IF (BORED(N).NE.SPACE) GO TO 20   !Is the space at the end?
      IF (ANY(BORED(1:N - 2) + 1 .NE. BORED(2:N - 1))) GO TO 20 !Are we there yet?
```

```
      WRITE (MSG,*) TRY,"Steps to success!" !Yes!
      END   !That was fun.
```

Output: Not so good. As ever, the character cell sizes are not square so a square game board comes out as a rectangle. Similarly, underlining is unavailable (no overprints!) so the layout is not pleasing. There are special "box-drawing" glyphs available, but they are not standardised and there is still no overprinting so that a flabby waste of space results. Further, there is no ability to re-write the display, even though one could easily regard the output to the screen as a random-access file: `WRITE (MSG,REC = 6) STUFF` would rewrite the sixth line of the display. Instead, output relentlessly rolls forwards, starting as follows:

```
To play 'slide-square' with 4 rows and 4 columns.
The game is to slide a square into the space
(thus leaving a space behind) until you attain
the nice orderly layout as follows:

Row|__1__2__3__4
  1|  1  2  3  4
  2|  5  6  7  8
  3|  9 10 11 12
  4| 13 14 15  0

But, your board looks like this...
Row|__1__2__3__4
  1| 15  0 14 11
  2|  8 13  5  3
  3|  4  1  7  9
  4| 10  6  2 12
Moveable pieces: 14,13,15. Choose one: 15

The new layout...
Row|__1__2__3__4
  1|  0 15 14 11
  2|  8 13  5  3
  3|  4  1  7  9
  4| 10  6  2 12
Moveable pieces: 15,8. Choose one:
```

The display here turns out to be less rectangular than that of the "console" screen's usual setting, which changes with the typeface and sizing anyway. Endless variation. As for playing the game, it is much easier to get a "feel" for the possibilities when manipulating the actual physical object. The digital world is less real.

FreeBASIC

```
sub drawboard( B() as ubyte )
    dim as string outstr = ""
    for i as ubyte = 0 to 15
        if B(i) = 0 then
            outstr = outstr + " XX "
        elseif B(i) < 10 then
            outstr = outstr + "  "+str(B(i))+" "
        else
            outstr = outstr + " " +str(B(i))+" "
        end if
        if i mod 4 = 3 then
            print outstr
            print
            outstr = ""
        end if
    next i
    print
end sub

function move( B() as ubyte, byref gap as ubyte, direction as ubyte ) as ubyte
    dim as integer targ = gap
    select case direction
```

```
        case 1  'UP
            targ = gap - 4
        case 2  'RIGHT
            if gap mod 4 = 3 then return gap
            targ = gap + 1
        case 3  'DOWN
            targ = gap + 4
        case 4
            if gap mod 4 = 0 then return gap
            targ = gap - 1
        case else
            return gap
    end select
    if targ > 15 or targ < 0 then return gap
    swap B(targ), B(gap)
    return targ
end function

sub shuffle( B() as ubyte, byref gap as ubyte )
    for i as ubyte = 0 to 100
        gap = move(B(), gap, int(rnd*4) + 1)
    next i
end sub

function solved( B() as ubyte ) as boolean
    dim as integer i
    for i = 0 to 14
        if B(i)>B(i+1) then return false
    next i
    return true
end function

dim as ubyte i, B(15), direction, gap
for i = 0 to 15
    B(i) = i
next i
shuffle B(), gap


while not solved( B() )
    cls
    drawboard B()
    print gap
    print "1 = up, 2=right, 3=down, 4=left"
    input direction
    gap = move( B(), gap, direction )
wend

cls
print "Congratulations! You win!"
print
drawboard B()
```

Gambas

```
'Charlie Ogier (C) 15PuzzleGame 24/04/2017 V0.1.0 Licenced under MIT
'Inspiration came from: -
''http://rosettacode.org/wiki/15_Puzzle_Game
''Bugs or comments to bugs@cogier.com
'Written in Gambas 3.9.2 - Updated on the Gambas Farm 01/05/2017
'Updated so that the message and the Title show the same amount of moves 01/06/2017
'Form now expandable. Font height automated. Form size and position saved 06/06/2107

'Simulate playing the 15 - game(puzzle)         Yes in GUI
'Generate a random start position               Yes
'Prompt the user for which piece To move        No
'Validate if the move is legal(possible)        Yes
'Display the game(puzzle) as pieces are moved   Yes in GUI
'Announce when the puzzle is solved             Yes
'Possibly keep track of the number of moves     Yes

byPos As New Byte[]                                         'Stores the position of the 'Tiles'
siMoves As Short                                            'Stores the amount of moves
hTimer As Timer                                             'Timer
```

```
dTimerStart As Date                                    'Stores the start time
dTimerDiff As Date                                     'Stores the time from the start to now
bTimerOn As Boolean                                    'To know if the Timer is running

Public Sub Form_Open()                                 'Form opens

Settings.read(Me, "Window")                            'Get details of the last window position and size
With Me                                                'With the Form..
  .Padding = 5                                         'Pad the edges
  .Arrangement = Arrange.Row                           'Arrange the Form
  .Title = "15PuzzleGame v0.3.0"                       'Set the Form Title
End With

BuildForm                                              'Go to the BuildForm routine

End

Public Sub BuildForm()                                 'To add items to the Form
Dim hButton As Button                                  'We need some Buttons
Dim byRand, byTest As Byte                             'Various variables
Dim bOK As Boolean                                     'Used to stop duplicate numbers being added
Dim bSolvable As Boolean

Repeat                                                 'Repeat until the puzzle is solvable
  Do                                                   'Start of a Do loop to create 0 to 15 in random
order
    byRand = Rand(0, 15)                               'Get a random number between 0 and 15
    If byRand = 0 Then byRand = 99                     'Change 0 to 99 for the Blank space
    bOK = True                                         'Set bOK to True
    For Each byTest In byPos                           'For each number stored in the array byPos
      If byRand = byTest Then bOK = False              'Check to see if it already exists, if it does set
bOK to False
    Next
    If bOK Then byPos.Add(byRand)                      'If not a duplicate then add it to the array
    If byPos.max = 15 Then Break                       'Once the array has 16 numbers get out of here. 99
is used for the blank space
  Loop
  bSolvable = IsItSolvable()                           'Go to the 'check if puzzle is solvable' routine
  If Not bSolvable Then byPos.clear                    'If it's not solvable then clear byPos
Until bSolvable = True                                 'Repeat until the puzzle is solvable

For byRand = 0 To 15                                   'Loop
  If byPos[byRand] = 99 Then                           'Check if value is 99 as this is where the blank
space will go
    AddPanel                                           'Go to the AddPanel routine to add the blank space
    Continue                                           'Skip to the end of the loop
  Endif
  hButton = New Button(Me) As "AllButtons"             'Add a new button to the Form, all buttons grouped
as 'AllButtons'
  With hButton                                         'With the following properties
    .Text = Str(byPos[byRand])                         'Add Button text
    .Tag = Str(byPos[byRand])                          'Add a Tag
    .Height = (Me.Height - 10) / 4                     'Set the Button height
    .Width = (Me.Width - 10) / 4                       'Set the Button width
    .Font.Bold = True                                  'Set the font to Bold
    .Font.Size = 16                                    'Set Font size
  End With
Next

AddTimer                                               'Go to the AddTimer routine

End


Public Sub AddPanel()                                  'Routine to add an invisible panel that is the
blank area
Dim hPanel As Panel                                    'We need a Panel

HPanel = New Panel(Me)                                 'Add the Panel to the Form
With HPanel                                            'With the following Properties
  .Tag = 99                                            'Set a Tag to 99
  .Height = (Me.Height - 10) / 4                       'Set the height
  .Width = (Me.Width - 10) / 4                         'Set the width
End With

End

Public Sub AddTimer()                                  'To add a Timer
```

```
hTimer = New Timer As "MyTimer"                    'Add a Timer
hTimer.Delay = 1000                                'Set the timer delay

End

Public Sub MyTimer_Timer()                         'Timer

Me.Title = siMoves & " Moves "                     'Set the Form Title to show the amount of moves
taken

If dTimerStart Then                                'If a start time has been set then
  dTimerDiff = Time(Now) - dTimerStart             'Calculate the time difference between StartTime
and Now
  Me.Title &= " - " & Str(dTimerDiff)              'Add the time taken to the Form Title
End If

End

Public Sub AllButtons_Click()                      'What to do when a Button is clicked
Dim byLast As Byte = Last.Tag                      'Get the Tag of the Button clicked
Dim byTemp, byCount As Byte                        'Various variables
Dim byCheck As Byte[] = [88, 88, 88, 88]           'Used for checking for the blank space
Dim byWChgeck As New Byte[16, 4]
Dim oObj As Object                                 'We need to enumerate Objects

For Each oObj In Me.Children                       'For each Object (Buttons in this case) that are
Children of the Form..
  If oObj.Tag = byLast Then Break                  'If the Tag of the Button matches then we know the
position of the Button on the form so get out of here
  Inc byCount                                      'Increase the value of byCount
Next

Select Case byCount                                'Depending on the position of the Button
  Case 0                                           'e.g 0 then we need to check positions 1 & 4 for
the blank
    byCheck[0] = 1
    byCheck[1] = 4
  Case 1
    byCheck[0] = 0
    byCheck[1] = 2
    byCheck[2] = 5
  Case 2
    byCheck[0] = 1
    byCheck[1] = 3
    byCheck[2] = 6
  Case 3
    byCheck[0] = 2
    byCheck[1] = 7
  Case 4
    byCheck[0] = 0
    byCheck[1] = 5
    byCheck[2] = 8
  Case 5                                           'e.g 5 then we need to check positions 1, 4, 6 & 9
for the blank
    byCheck[0] = 1
    byCheck[1] = 4
    byCheck[2] = 6
    byCheck[3] = 9
  Case 6
    byCheck[0] = 2
    byCheck[1] = 5
    byCheck[2] = 7
    byCheck[3] = 10
  Case 7
    byCheck[0] = 3
    byCheck[1] = 6
    byCheck[2] = 11
  Case 8
    byCheck[0] = 4
    byCheck[1] = 9
    byCheck[2] = 12
  Case 9
    byCheck[0] = 5
    byCheck[1] = 8
    byCheck[2] = 10
    byCheck[3] = 13
  Case 10
```

```
    byCheck[0] = 6
    byCheck[1] = 9
    byCheck[2] = 11
    byCheck[3] = 14
  Case 11
    byCheck[0] = 7
    byCheck[1] = 10
    byCheck[2] = 15
  Case 12
    byCheck[0] = 8
    byCheck[1] = 13
  Case 13
    byCheck[0] = 9
    byCheck[1] = 12
    byCheck[2] = 14
  Case 14
    byCheck[0] = 10
    byCheck[1] = 13
    byCheck[2] = 15
  Case 15
    byCheck[0] = 11
    byCheck[1] = 14
End Select

For Each byTemp In byCheck                                  'For each value in byCheck
  If byTemp = 88 Then Break                                 'If byTemp = 88 then get out of here
  If byPos[byTemp] = 99 Then                                'If the position checked is 99 (the blank) then..
    byPos[byTemp] = Last.Text                               'Set the new position of the Tile in byPos
    byPos[byCount] = 99                                     'Set the existing Tile position to = 99 (blank)
    Inc siMoves                                             'Inc the amount of moves made
    If Not bTimerOn Then                                    'If the Timer is now needed then
      dTimerStart = Time(Now)                               'Set the Start time to NOW
      hTimer.start                                          'Start the Timer
      bTimerOn = True                                       'Set bTimerOn to True
    Endif
    Break                                                   'Get out of here
  End If
Next

RebuildForm                                                 'Go to the RebuilForm routine
CheckIfPuzzleCompleted                                      'Check to see if the puzzle has been solved

End

Public Sub CheckIfPuzzleCompleted()                         'Is the puzzle is complete
Dim byTest As Byte[] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 99] 'byPos will equal this if it is
completed
Dim siCount As Short                                        'Counter
Dim bCompleted As Boolean = True                            'Completed?
Dim sMessage As String                                      'String to store the display message

For siCount = 0 To 15                                       'Loop through the byPos
  If byPos[siCount] <> byTest[siCount] Then                 'If the position does not match the completed
position then..
    bCompleted = False                                      'Set bCompleted to False
    Break                                                   'Get out of here
  Endif
Next

If bCompleted Then                                          'If the puzzle is completed then
  hTimer.Stop                                               'Stop the timer
  Me.Title = siMoves & " Moves "                            'Set the Form Title to show the amount of moves
taken
  sMessage = "Congratulations!!\n"                          'Build sMessage
  sMessage &= Str(siMoves) & " Moves\n"                     'Build sMessage
  sMessage &= "Time = " & Str(dTimerDiff)                   'Build sMessage
  Message(sMessage, "OK")                                   'Put up a congratulatory message
  Me.Close                                                  'Close the form
Endif

End

Public Sub RebuildForm()                                    'To clear the Form and rebuild with the Tiles in
the new postion
Dim hButton As Button                                       'We need Buttons
Dim byCount, byTemp As Byte                                 'Various variables
Dim siFontH As Short
```

```
Me.Children.clear                                                   'Clear the Form of all Objects

For Each byTemp In byPos                                            'For each 'Position'
  If byTemp = 99 Then                                               'If the Position's value is 99 then it's the space
    AddPanel                                                        'Go to the AddPanel routine
  Else                                                              'If the Position's value is NOT 99 then
    hButton = New Button(Me) As "AllButtons"                        'Create a new Button
  With hButton                                                      'With the following properties
    .Text = Str(byPos[byCount])                                     'Text as stored in byPos
    .Tag = Str(byPos[byCount])                                      'Tag as stored in byPos
    .Height = (Me.Height - 10) / 4                                  'Set the Button height
    .Width = (Me.Width - 10) / 4                                    'Set the Button width
    .Font.Bold = True                                               'Set the Font to Bold
      End With
    If Me.Width > Me.Height Then                                    'If Form Width is greater than Form Width then..
      siFontH = Me.Height                                           'siFontH = Form Height
    Else                                                            'Else..
      siFontH = Me.Width                                            'siFontH = Form Width
    End If
  hButton.Font.size = siFontH / 16                                  'Set Font height
  Endif

  Inc byCount                                                       'Increase counter
Next

End

Public Sub Form_Resize()                                            'If the form is resized

RebuildForm                                                         'Rebuild the Form

End

Public Sub IsItSolvable() As Boolean                                'To check if the puzzle is solvable
Dim bSolvable, bBlankOnEven As Boolean                              'Triggers
Dim siCount0, siCount1, siInversion As Short                        'Counters

For siCount0 = 0 To byPos.Max                                       'Loop through the positions
  If byPos[siCount0] = 99 Then                                      'The blank
    If InStr("0,1,2,3,8,9,10,11,", Str(siCount0 & ",")) Then        'Is the blank on an even row (counting from the
bottom) if so..
      bBlankOnEven = True                                           'bBlankOnEven = True
    End If
    Continue                                                        'Go to the end of the loop
  End If
  For siCount1 = siCount0 + 1 To byPos.Max                          'Loop through the positions
    If byPos[siCount0] > byPos[siCount1] Then Inc siInversion       'Counts the inversions
  Next                                                              'See
https://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html
Next

If bBlankOnEven And Odd(siInversion) Then bSolvable = True          'Blank is on an even row (counting from the
bottom) then the number of inversions in a solvable situation is odd
If Not bBlankOnEven And Even(siInversion) Then bSolvable = True 'Blank is on an odd row (counting from the bottom)
then the number of inversions in a solvable situation is even

Return bSolvable                                                    'Return the value

End

Public Sub Form_Close()

Settings.Write(Me, "Window")                                        'Store the window position and size

End
```

Click here for image of game in play (http://www.cogier.com/gambas/Copuzzle.png)

Go

```
package main

import (
    "fmt"
```

```
    "math/rand"
    "strings"
    "time"
)

func main() {
    rand.Seed(time.Now().UnixNano())
    p := newPuzzle()
    p.play()
}

type board [16]cell
type cell uint8
type move uint8

const (
    up move = iota
    down
    right
    left
)

func randMove() move { return move(rand.Intn(4)) }

var solvedBoard = board{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0}

func (b *board) String() string {
    var buf strings.Builder
    for i, c := range b {
        if c == 0 {
            buf.WriteString("  .")
        } else {
            _, _ = fmt.Fprintf(&buf, "%3d", c)
        }
        if i%4 == 3 {
            buf.WriteString("\n")
        }
    }
    return buf.String()
}

type puzzle struct {
    board board
    empty int // board[empty] == 0
    moves int
    quit  bool
}

func newPuzzle() *puzzle {
    p := &puzzle{
        board: solvedBoard,
        empty: 15,
    }
    // Could make this configurable, 10==easy, 50==normal, 100==hard
    p.shuffle(50)
    return p
}

func (p *puzzle) shuffle(moves int) {
    // As with other Rosetta solutions, we use some number
    // of random moves to "shuffle" the board.
    for i := 0; i < moves || p.board == solvedBoard; {
        if p.doMove(randMove()) {
            i++
        }
    }
}

func (p *puzzle) isValidMove(m move) (newIndex int, ok bool) {
    switch m {
    case up:
        return p.empty - 4, p.empty/4 > 0
    case down:
        return p.empty + 4, p.empty/4 < 3
    case right:
        return p.empty + 1, p.empty%4 < 3
    case left:
        return p.empty - 1, p.empty%4 > 0
```

```
    default:
        panic("not reached")
    }
}

func (p *puzzle) doMove(m move) bool {
    i := p.empty
    j, ok := p.isValidMove(m)
    if ok {
        p.board[i], p.board[j] = p.board[j], p.board[i]
        p.empty = j
        p.moves++
    }
    return ok
}

func (p *puzzle) play() {
    fmt.Printf("Starting board:")
    for p.board != solvedBoard && !p.quit {
        fmt.Printf("\n%v\n", &p.board)
        p.playOneMove()
    }
    if p.board == solvedBoard {
        fmt.Printf("You solved the puzzle in %d moves.\n", p.moves)
    }
}

func (p *puzzle) playOneMove() {
    for {
        fmt.Printf("Enter move #%d (U, D, L, R, or Q): ", p.moves+1)
        var s string
        if n, err := fmt.Scanln(&s); err != nil || n != 1 {
            continue
        }

        s = strings.TrimSpace(s)
        if s == "" {
            continue
        }

        var m move
        switch s[0] {
        case 'U', 'u':
            m = up
        case 'D', 'd':
            m = down
        case 'L', 'l':
            m = left
        case 'R', 'r':
            m = right
        case 'Q', 'q':
            fmt.Printf("Quiting after %d moves.\n", p.moves)
            p.quit = true
            return
        default:
            fmt.Println(`
Please enter "U", "D", "L", or "R" to move the empty cell
up, down, left, or right. You can also enter "Q" to quit.
Upper or lowercase is accepted and only the first non-blank
character is important (i.e. you may enter "up" if you like).
`)
            continue
        }

        if !p.doMove(m) {
            fmt.Println("That is not a valid move at the moment.")
            continue
        }

        return
    }
}
```

Harbour

```
#include "inkey.ch"
#include "Box.ch"


procedure Main()
    // console init
    SET SCOREBOARD OFF
    SetMode(30,80)
    ret := 0

    // main loop
    yn := .F.
    DO WHILE yn == .F.
        // draw console
        cls
        @ 0, 0 TO MaxRow(), MaxCol() DOUBLE
        SetColor("BG+/B,W/N")
        @ 0, 4 SAY "  Slidng puzzle game  "
        SetColor()

        // input size of grid
        tam := 0
        @ MaxRow() - 2, 4 SAY "Size of grid: " GET tam PICTURE "9"
        READ

        // Initialize numbers
        lista := ARRAY (tam * tam)
        FOR z := 1 TO tam * tam
            lista[z] := z
        NEXT
        lista1 := lista
        grid := ARRAY (tam,tam)

        // populate grid with numbers
        FOR i := 1 TO tam
            FOR j := 1 TO tam
                grid[i,j] := lista1[ (i-1) * tam + j]
            NEXT
        NEXT
        Mostra(@grid)
        InKey(0)

        // initialize the game
        n := 0
        t := 0
        lista := lista1     // lista for scrambling, lista1 preserve numbers in order
        DO WHILE .T.
            // scrambling numbers
            FOR i := 1 TO tam*tam
                n := Int ( ft_Rand1(tam * tam - 1) + 1 )
                t := lista[n]
                lista[n] := lista[i]
                lista[i] := t
            NEXT
            // have solution?
            possp := 0
            invct := 0  // change counter
            FOR i := 1 TO tam * tam -1
                IF lista[i] != tam*tam
                    FOR j := i + 1 TO tam * tam
                        IF lista[j] != tam*tam
                            IF lista[i] > lista[j]
                                invct++
                            ENDIF
                        ENDIF
                    NEXT
                ELSE
                    possp := i
                ENDIF
            NEXT
            linv := If( ( (invct % 2) == 0 ), .T., .F.)
            lkin := If( ( (tam - Int( (possp -1) / tam )) % 2) == 0, .T., .F. )

            IF ( (tam % 2) != 0)    // if grid size is odd
                IF linv                 // if number of positions changes is even, solvable
                    EXIT
```

```
            ELSE
                LOOP                // if is odd, not solvable, scramble more
            ENDIF               // if grid size is even
        ELSE
                                    // If changes is even and space position is in odd line
                                    // or changes is odd and space position is in even line
                                    // (XOR condition) is solvable
            IF (linv .AND. !lkin) .OR. (!linv .AND. lkin) // XOR !!!
                EXIT
            ElSE                    // else scramble more
                LOOP
            ENDIF
        ENDIF

    ENDDO

    // populate the grid with scrambled numbers
    FOR i := 1 TO tam
        FOR j := 1 TO tam
            grid[i,j] := lista[ (i-1) * tam + j]
        NEXT
    NEXT
    ret := Mostra(@grid)

// play
    key := 0
    DO WHILE LastKey() != K_ESC
        key := 0
        // find the space coords
        xe := 0
        ye := 0
        lv := tam*tam
        FOR i := 1 TO tam
            FOR j := 1 TO tam
                IF grid[i,j] == lv
                    xe :=i
                    ye :=j
                ENDIF
            NEXT
        NEXT
        // the direction keys
        key := inkey(0)
        DO CASE
            CASE key == K_UP
                IF xe > 1
                    grid[xe,ye] := grid[xe-1,ye]
                    grid[xe-1,ye] := lv
                ENDIF
                ret := Mostra(@grid)
            CASE key == K_DOWN
                IF xe < tam
                    grid[xe,ye] := grid[xe+1,ye]
                    grid[xe+1,ye] := lv
                ENDIF
                ret := Mostra(@grid)
            CASE key == K_LEFT
                IF ye > 1
                    grid[xe,ye] := grid[xe,ye-1]
                    grid[xe,ye-1] := lv
                ENDIF
                ret := Mostra(@grid)
            CASE key == K_RIGHT
                IF ye < tam
                    grid[xe,ye] := grid[xe,ye+1]
                    grid[xe,ye+1] := lv
                ENDIF
                ret := Mostra(@grid)
        ENDCASE
        IF ret == tam*tam-1                             // ret is qtty numbers in position
            @ MaxRow() - 3, 4 SAY "Fim de jogo!"        // if ret == (size*size) -1
            key := K_ESC                                // all numbers in position
            EXIT                                        // game solved
        ENDIF
    ENDDO
    @ MaxRow() - 2, 4 SAY "Deseja sair? (yn): " GET yn PICTURE "Y"
    READ
    @ MaxRow() - 3, 4 SAY "              "
ENDDO
```

```
return NIL

FUNCTION Mostra(grid)
    // Show the gris
    fim := 0                                                    // how many numbers in position?
    SetColor("BG+/B,W/N")
    @ 5,10 , 5 + tam * 2, 9 + tam * 4 BOX B_SINGLE + Space(1)
    i := 0
    FOR y := 1 TO tam
        FOR x := 1 TO tam
            IF grid[x,y] == tam * tam                           // show space
                SetColor(" B/GR+, W/N")
                @ x*2 + 4, i + 11 SAY "  "
                SetColor("BG+/B,W/N")
            ELSE
                IF ( (x-1) * tam + y ) == grid[x,y]             // show number in position
                    SetColor("W/G,W/N")
                    @ x*2 + 4, i + 11 SAY grid[x,y] PICTURE "99"
                    fim++
                ELSE                                            // show number out position
                    SetColor("BG+/B,W/N")
                    @ x*2 + 4, i + 11 SAY grid[x,y] PICTURE "99"
                ENDIF
            ENDIF
        NEXT
        i = i + 4
    NEXT
    SetColor(" W/N, BG+/B")
RETURN fim
```

Haskell

```haskell
import Data.Array
import System.Random

type Puzzle = Array (Int, Int) Int

main :: IO ()
main = do
    putStrLn "Please enter the difficulty level: 0, 1 or 2"
    userInput <- getLine
    let diffLevel = read userInput
    if userInput == "" || any (\c -> c < '0' || c > '9') userInput || diffLevel > 2 || diffLevel < 0
        then putStrLn "That is not a valid difficulty level." >> main
        else shufflePuzzle ([10, 50, 100] !! diffLevel) solvedPuzzle >>= gameLoop

gameLoop :: Puzzle -> IO ()
gameLoop puzzle
    | puzzle == solvedPuzzle = putStrLn "You solved the puzzle!" >> printPuzzle puzzle
    | otherwise = do
    printPuzzle puzzle
    putStrLn "Please enter number to move"
    userInput <- getLine
    if any (\c -> c < '0' || c > '9') userInput
        then putStrLn "That is not a valid number." >> gameLoop puzzle
        else let move = read userInput in
            if move `notElem` validMoves puzzle
                then putStrLn "This move is not available." >> gameLoop puzzle
                else gameLoop (applyMove move puzzle)

validMoves :: Puzzle -> [Int]
validMoves puzzle = [puzzle ! (row', column') |
                     row' <- [rowEmpty-1..rowEmpty+1], column' <- [columnEmpty-1..columnEmpty+1],
                     row' < 4, row' >= 0, column' < 4, column' >= 0,
                     (row' == rowEmpty) /= (column' == columnEmpty)]
    where (rowEmpty, columnEmpty) = findIndexOfNumber 16 puzzle

applyMove :: Int -> Puzzle -> Puzzle
applyMove numberToMove puzzle = puzzle // [(indexToMove, 16), (emptyIndex, numberToMove)]
    where indexToMove = findIndexOfNumber numberToMove puzzle
          emptyIndex = findIndexOfNumber 16 puzzle

findIndexOfNumber :: Int -> Puzzle -> (Int, Int)
findIndexOfNumber number puzzle = case filter (\idx -> number == puzzle ! idx)
```

```
                                        (indices puzzle) of
                                    [idx] -> idx
                                    _ -> error "BUG: number not in puzzle"

printPuzzle :: Puzzle -> IO ()
printPuzzle puzzle = do
    putStrLn "+--+--+--+--+"
    putStrLn $ "|" ++ formatCell (0, 0) ++ "|" ++ formatCell (0, 1) ++ "|" ++ formatCell (0, 2) ++ "|" ++
formatCell (0, 3) ++ "|"
    putStrLn "+--+--+--+--+"
    putStrLn $ "|" ++ formatCell (1, 0) ++ "|" ++ formatCell (1, 1) ++ "|" ++ formatCell (1, 2) ++ "|" ++
formatCell (1, 3) ++ "|"
    putStrLn "+--+--+--+--+"
    putStrLn $ "|" ++ formatCell (2, 0) ++ "|" ++ formatCell (2, 1) ++ "|" ++ formatCell (2, 2) ++ "|" ++
formatCell (2, 3) ++ "|"
    putStrLn "+--+--+--+--+"
    putStrLn $ "|" ++ formatCell (3, 0) ++ "|" ++ formatCell (3, 1) ++ "|" ++ formatCell (3, 2) ++ "|" ++
formatCell (3, 3) ++ "|"
    putStrLn "+--+--+--+--+"
    where formatCell idx
              | i == 16 = "  "
              | i > 9 = show i
              | otherwise = " " ++ show i
              where i = puzzle ! idx

solvedPuzzle :: Puzzle
solvedPuzzle = listArray ((0, 0), (3, 3)) [1..16]

shufflePuzzle :: Int -> Puzzle -> IO Puzzle
shufflePuzzle 0 puzzle = return puzzle
shufflePuzzle numOfShuffels puzzle = do
    let moves = validMoves puzzle
    randomIndex <- randomRIO (0, length moves - 1)
    let move = moves !! randomIndex
    shufflePuzzle (numOfShuffels - 1) (applyMove move puzzle)
```

Output:

```
Please enter the difficulty level: 0, 1 or 2
0
+--+--+--+--+
| 1| 6| 2| 4|
+--+--+--+--+
| 5|10| 3| 8|
+--+--+--+--+
| 9|14| 7|11|
+--+--+--+--+
|13|  |15|12|
+--+--+--+--+
Please enter number to move
14
+--+--+--+--+
| 1| 6| 2| 4|
+--+--+--+--+
| 5|10| 3| 8|
+--+--+--+--+
| 9|  | 7|11|
+--+--+--+--+
|13|14|15|12|
+--+--+--+--+
Please enter number to move
```

J

Implementation:

```
require'general/misc/prompt'

genboard=:3 :0
  b=. ?~16
  if. 0<C.!.2 b do.
```

```
    b=. (<0 _1)C. b
  end.
  a: (b i.0)} <"0 b
)

done=: (<"0]1+i.15),a:

shift=: |.!._"0 2
taxi=: |:,/"2(_1 1 shift i.4 4),_1 1 shift"0 1/ i.4 4

showboard=:3 :0
  echo 'current board:'
  echo 4 4$y
)

help=:0 :0

  Slide a number block into the empty space
  until you get:
┌──┬──┬──┬──┐
│1 │2 │3 │4 │
├──┼──┼──┼──┤
│5 │6 │7 │8 │
├──┼──┼──┼──┤
│9 │10│11│12│
├──┼──┼──┼──┤
│13│14│15│  │
└──┴──┴──┴──┘
  Or type 'q' to quit.
)

getmove=:3 :0
  showboard y
  blank=. y i. a:
  options=. /:~ ;y {~ _ -.~ blank { taxi
  whilst. -. n e. options do.
    echo 'Valid moves: ',":options
    t=. prompt 'move which number? '
    if. 'q' e. t do.
      echo 'giving up'
      throw.
    elseif. 'h' e. t do.
      echo help
      showboard y
    end.
    n=. {._".t
  end.
  (<blank,y i.<n) C. y
)

game=: 3 :0
  echo '15 puzzle'
  echo 'h for help, q to quit'
  board=. genboard''
  whilst. -. done-:board do.
    board=. getmove board
  end.
  showboard board
  echo 'You win.'
)
```

Most of this is user interface code. We initially shuffle the numbers randomly, then check their parity and swap the first and last squares if needed. Then, for each move, we allow the user to pick one of the taxicab neighbors of the empty square.

A full game would be too big to be worth showing here, so for the purpose of giving a glimpse of what this looks like in action we replace the random number generator with a constant:

```
   game''
15 puzzle
h for help, q to quit
current board:
┌──┬──┬──┬──┐
```

```
+--+--+--+--+
| 1| 2| 3| 4|
+--+--+--+--+
| 5| 6| 7| 8|
+--+--+--+--+
| 9|10|  |11|
+--+--+--+--+
|13|14|15|12|
+--+--+--+--+
Valid moves: 7 10 11 15
move which number? 11
current board:
+--+--+--+--+
| 1| 2| 3| 4|
+--+--+--+--+
| 5| 6| 7| 8|
+--+--+--+--+
| 9|10|11|  |
+--+--+--+--+
|13|14|15|12|
+--+--+--+--+
Valid moves: 8 11 12
move which number? 12
current board:
+--+--+--+--+
| 1| 2| 3| 4|
+--+--+--+--+
| 5| 6| 7| 8|
+--+--+--+--+
| 9|10|11|12|
+--+--+--+--+
|13|14|15|  |
+--+--+--+--+
You win.
```

Java

Works with: Java version 8

```
package fifteenpuzzle;

import java.awt.*;
import java.awt.event.*;
import java.util.Random;
import javax.swing.*;

class FifteenPuzzle extends JPanel {

    private final int side = 4;
    private final int numTiles = side * side - 1;

    private final Random rand = new Random();
    private final int[] tiles = new int[numTiles + 1];
    private final int tileSize;
    private int blankPos;
    private final int margin;
    private final int gridSize;
    private boolean gameOver;

    private FifteenPuzzle() {
        final int dim = 640;

        margin = 80;
        tileSize = (dim - 2 * margin) / side;
        gridSize = tileSize * side;

        setPreferredSize(new Dimension(dim, dim + margin));
        setBackground(Color.WHITE);
        setForeground(new Color(0x6495ED)); // cornflowerblue
        setFont(new Font("SansSerif", Font.BOLD, 60));

        gameOver = true;

        addMouseListener(new MouseAdapter() {
            @Override
```

```
            public void mousePressed(MouseEvent e) {
                if (gameOver) {
                    newGame();

                } else {

                    int ex = e.getX() - margin;
                    int ey = e.getY() - margin;

                    if (ex < 0 || ex > gridSize || ey < 0 || ey > gridSize) {
                        return;
                    }

                    int c1 = ex / tileSize;
                    int r1 = ey / tileSize;
                    int c2 = blankPos % side;
                    int r2 = blankPos / side;

                    int clickPos = r1 * side + c1;

                    int dir = 0;
                    if (c1 == c2 && Math.abs(r1 - r2) > 0) {
                        dir = (r1 - r2) > 0 ? 4 : -4;

                    } else if (r1 == r2 && Math.abs(c1 - c2) > 0) {
                        dir = (c1 - c2) > 0 ? 1 : -1;
                    }

                    if (dir != 0) {
                        do {
                            int newBlankPos = blankPos + dir;
                            tiles[blankPos] = tiles[newBlankPos];
                            blankPos = newBlankPos;
                        } while (blankPos != clickPos);
                        tiles[blankPos] = 0;
                    }

                    gameOver = isSolved();
                }
                repaint();
            }
        });

        newGame();
    }

    private void newGame() {
        do {
            reset();
            shuffle();
        } while (!isSolvable());
        gameOver = false;
    }

    private void reset() {
        for (int i = 0; i < tiles.length; i++) {
            tiles[i] = (i + 1) % tiles.length;
        }
        blankPos = tiles.length - 1;
    }

    private void shuffle() {
        // don't include the blank space in the shuffle, leave it
        // in the home position
        int n = numTiles;
        while (n > 1) {
            int r = rand.nextInt(n--);
            int tmp = tiles[r];
            tiles[r] = tiles[n];
            tiles[n] = tmp;
        }
    }

    /*  Only half the permutations of the puzzle are solvable.

        Whenever a tile is preceded by a tile with higher value it counts
        as an inversion. In our case, with the blank space in the home
        position, the number of inversions must be even for the puzzle
```

```
        to be solvable.

        See also:
        www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html
     */
    private boolean isSolvable() {
        int countInversions = 0;
        for (int i = 0; i < numTiles; i++) {
            for (int j = 0; j < i; j++) {
                if (tiles[j] > tiles[i]) {
                    countInversions++;
                }
            }
        }
        return countInversions % 2 == 0;
    }

    private boolean isSolved() {
        if (tiles[tiles.length - 1] != 0) {
            return false;
        }
        for (int i = numTiles - 1; i >= 0; i--) {
            if (tiles[i] != i + 1) {
                return false;
            }
        }
        return true;
    }

    private void drawGrid(Graphics2D g) {
        for (int i = 0; i < tiles.length; i++) {
            int r = i / side;
            int c = i % side;
            int x = margin + c * tileSize;
            int y = margin + r * tileSize;

            if (tiles[i] == 0) {
                if (gameOver) {
                    g.setColor(Color.GREEN);
                    drawCenteredString(g, "\u2713", x, y);
                }
                continue;
            }

            g.setColor(getForeground());
            g.fillRoundRect(x, y, tileSize, tileSize, 25, 25);
            g.setColor(Color.blue.darker());
            g.drawRoundRect(x, y, tileSize, tileSize, 25, 25);
            g.setColor(Color.WHITE);

            drawCenteredString(g, String.valueOf(tiles[i]), x, y);
        }
    }

    private void drawStartMessage(Graphics2D g) {
        if (gameOver) {
            g.setFont(getFont().deriveFont(Font.BOLD, 18));
            g.setColor(getForeground());
            String s = "click to start a new game";
            int x = (getWidth() - g.getFontMetrics().stringWidth(s)) / 2;
            int y = getHeight() - margin;
            g.drawString(s, x, y);
        }
    }

    private void drawCenteredString(Graphics2D g, String s, int x, int y) {
        FontMetrics fm = g.getFontMetrics();
        int asc = fm.getAscent();
        int des = fm.getDescent();

        x = x + (tileSize - fm.stringWidth(s)) / 2;
        y = y + (asc + (tileSize - (asc + des)) / 2);

        g.drawString(s, x, y);
    }

    @Override
    public void paintComponent(Graphics gg) {
```

```
        super.paintComponent(gg);
        Graphics2D g = (Graphics2D) gg;
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);

        drawGrid(g);
        drawStartMessage(g);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame f = new JFrame();
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.setTitle("Fifteen Puzzle");
            f.setResizable(false);
            f.add(new FifteenPuzzle(), BorderLayout.CENTER);
            f.pack();
            f.setLocationRelativeTo(null);
            f.setVisible(true);
        });
    }
}
```

JavaScript

Play it here (http://paulo-jorente.de/webgames/15p/)

```
var board, zx, zy, clicks, possibles, clickCounter, oldzx = -1, oldzy = -1;
function getPossibles() {
    var ii, jj, cx = [-1, 0, 1, 0], cy = [0, -1, 0, 1];
    possibles = [];
    for( var i = 0; i < 4; i++ ) {
        ii = zx + cx[i]; jj = zy + cy[i];
        if( ii < 0 || ii > 3 || jj < 0 || jj > 3 ) continue;
        possibles.push( { x: ii, y: jj } );
    }
}
function updateBtns() {
    var b, v, id;
    for( var j = 0; j < 4; j++ ) {
        for( var i = 0; i < 4; i++ ) {
            id = "btn" + ( i + j * 4 );
            b = document.getElementById( id );
            v = board[i][j];
            if( v < 16 ) {
                b.innerHTML = ( "" + v );
                b.className = "button"
            }
            else {
                b.innerHTML = ( "" );
                b.className = "empty";
            }
        }
    }
    clickCounter.innerHTML = "Clicks: " + clicks;
}
function shuffle() {
    var v = 0, t;
    do {
        getPossibles();
        while( true ) {
            t = possibles[Math.floor( Math.random() * possibles.length )];
            console.log( t.x, oldzx, t.y, oldzy )
            if( t.x != oldzx || t.y != oldzy ) break;
        }
        oldzx = zx; oldzy = zy;
        board[zx][zy] = board[t.x][t.y];
        zx = t.x; zy = t.y;
        board[zx][zy] = 16;
    } while( ++v < 200 );
}
function restart() {
    shuffle();
```

```
    clicks = 0;
    updateBtns();
}
function checkFinished() {
    var a = 0;
    for( var j = 0; j < 4; j++ ) {
        for( var i = 0; i < 4; i++ ) {
            if( board[i][j] < a ) return false;
            a = board[i][j];
        }
    }
    return true;
}
function btnHandle( e ) {
    getPossibles();
    var c = e.target.i, r = e.target.j, p = -1;
    for( var i = 0; i < possibles.length; i++ ) {
        if( possibles[i].x == c && possibles[i].y == r ) {
            p = i;
            break;
        }
    }
    if( p > -1 ) {
        clicks++;
        var t = possibles[p];
        board[zx][zy] = board[t.x][t.y];
        zx = t.x; zy = t.y;
        board[zx][zy] = 16;
        updateBtns();
        if( checkFinished() ) {
            setTimeout(function(){
                alert( "WELL DONE!" );
                restart();
            }, 1);
        }
    }
}
function createBoard() {
    board = new Array( 4 );
    for( var i = 0; i < 4; i++ ) {
        board[i] = new Array( 4 );
    }
    for( var j = 0; j < 4; j++ ) {
        for( var i = 0; i < 4; i++ ) {
            board[i][j] = ( i + j * 4 ) + 1;
        }
    }
    zx = zy = 3; board[zx][zy] = 16;
}
function createBtns() {
    var b, d = document.createElement( "div" );
    d.className += "board";
    document.body.appendChild( d );
    for( var j = 0; j < 4; j++ ) {
        for( var i = 0; i < 4; i++ ) {
            b = document.createElement( "button" );
            b.id = "btn" + ( i + j * 4 );
            b.i = i; b.j = j;
            b.addEventListener( "click", btnHandle, false );
            b.appendChild( document.createTextNode( "" ) );
            d.appendChild( b );
        }
    }
    clickCounter = document.createElement( "p" );
    clickCounter.className += "txt";
    document.body.appendChild( clickCounter );
}
function start() {
    createBtns();
    createBoard();
    restart();
}
```

Html to test

```
<!DOCTYPE html>
<html><head><meta charset="UTF-8">
<title>15 Puzzle</title>
<script src="p15.js"></script>
<style>
    html,body{padding:0; margin:0;padding-top:8vh;background:#222;color:#111}
    .txt{color:#fff;text-align:center;font-size:5vh}
    .board{padding:0;margin:auto;width:33vh;height:33vh}
    .button, .empty{border:0;font-size:3.5vh;margin:0.5vh;padding:0;height:6vh;width:7.25vh;line-height:5vh;
    vertical-align:middle;background:#fff;text-align:center;border-radius:3px;cursor:pointer;float:left}
    .empty{background:#333;border:1px solid #111}
</style>
</head><body onload="start()"></body></html>
```

Julia

```
using Random

const size = 4
const puzzle = string.(reshape(1:16, size, size))
puzzle[16] = " "
rng = MersenneTwister(Int64(round(time())))
shufflepuzzle() = (puzzle .= shuffle(rng, puzzle))
findtile(t) = findfirst(x->x == t, puzzle)
findhole() = findtile(" ")

function issolvable()
    inversioncount = 1
    asint(x) = (puzzle[x] == " ") ? 0 : parse(Int64, puzzle[x])
    for i in 1:size^2-1, j in i:size^2
        if puzzle[i] == " " || puzzle[j] == " "
            continue
        end
        if parse(Int, puzzle[i]) < parse(Int, puzzle[j])
            inversioncount += 1
        end
    end
    if size % 2 == 1
        return inversioncount % 2 == 0
    end
    pos = findhole()
    inversioncount += pos[2]
    return inversioncount & 1 == 0
end

function nexttohole()
    holepos = findhole()
    row = holepos[1]
    col = holepos[2]
    if row == 1
        if col == 1
            return [[row, col + 1], [row + 1, col]]
        elseif col == size
            return [[row, col - 1], [row + 1, col]]
        else
            return [[row, col - 1], [row, col + 1], [row + 1, col]]
        end
    elseif row == size
        if col == 1
            return [[row - 1, col], [row, col + 1]]
        elseif col == size
            return [[row - 1, col], [row, col - 1]]
        else
            return [[row - 1, col], [row, col - 1], [row, col + 1]]
        end
    else
        if col == 1
            return [[row - 1, col], [row, col + 1], [row + 1, col]]
        elseif col == size
            return [[row - 1, col], [row, col - 1], [row + 1, col]]
        else
            return [[row - 1, col], [row, col - 1], [row, col + 1], [row + 1, col]]
        end
```

```
    end
end

possiblemoves() = map(pos->puzzle[pos[1], pos[2]], nexttohole())

function movehole(tiletofill)
    if tiletofill in possiblemoves()
        curpos = findtile(tiletofill)
        holepos = findhole()
        puzzle[holepos] = tiletofill
        puzzle[curpos] = " "
    else
        println("Bad tile move $tiletofill.\nPossible moves are $(possiblemoves()).")
    end
end

function printboard()
    ppuz(x,y) = print(lpad(rpad(puzzle[x,y], 3), 4), "|")
    print("+----+----+----+----+\n|")
    for j in 1:size, i in 1:size
        ppuz(i,j)
        if i == size
            print("\n")
            if j < size
                print("|")
            end
        end
    end
    println("+----+----+----+----+")

end

function puzzle15play()
    solved() = (puzzle[1:15] == string.(1:15))
    shufflepuzzle()
    println("This puzzle is", issolvable() ? " " : " not ", "solvable.")
    while !solved()
        printboard()
        print("Possible moves are: $(possiblemoves()), 0 to exit. Your move? =>  ")
        s = readline()
        if s == "0"
            exit(0)
        end
        movehole(s)
    end
    printboard()
    println("Puzzle solved.")
end

puzzle15play()
```

Output:

```
This puzzle is solvable.
+----+----+----+----+
| 5  | 7  | 14 |    |
| 4  | 3  | 13 | 12 |
| 10 | 1  | 6  | 9  |
| 8  | 15 | 11 | 2  |
+----+----+----+----+
Possible moves are: ["14", "12"], 0 to exit. Your move? =>  12
+----+----+----+----+
| 5  | 7  | 14 | 12 |
| 4  | 3  | 13 |    |
| 10 | 1  | 6  | 9  |
```

```
| 8  | 15 | 11 | 2  |
+----+----+----+----+
Possible moves are: ["13", "12", "9"], 0 to exit. Your move? =>
```

Kotlin

Translation of: Java

```
// version 1.1.3

import java.awt.BorderLayout
import java.awt.Color
import java.awt.Dimension
import java.awt.Font
import java.awt.Graphics
import java.awt.Graphics2D
import java.awt.RenderingHints
import java.awt.event.MouseAdapter
import java.awt.event.MouseEvent
import java.util.Random
import javax.swing.JFrame
import javax.swing.JPanel
import javax.swing.SwingUtilities

class FifteenPuzzle(dim: Int, val margin: Int) : JPanel() {

    private val rand = Random()
    private val tiles = IntArray(16)
    private val tileSize = (dim - 2 * margin) / 4
    private val gridSize = tileSize * 4
    private var blankPos = 0

    init {
        preferredSize = Dimension(dim, dim)
        background = Color.white
        val cornflowerBlue = 0x6495ED
        foreground = Color(cornflowerBlue)
        font = Font("SansSerif", Font.BOLD, 60)

        addMouseListener(object : MouseAdapter() {
            override fun mousePressed(e: MouseEvent) {
                val ex = e.x - margin
                val ey = e.y - margin
                if (ex !in 0..gridSize || ey !in 0..gridSize) return

                val c1 = ex / tileSize
                val r1 = ey / tileSize
                val c2 = blankPos % 4
                val r2 = blankPos / 4
                if ((c1 == c2 && Math.abs(r1 - r2) == 1) ||
                    (r1 == r2 && Math.abs(c1 - c2) == 1)) {
                    val clickPos = r1 * 4 + c1
                    tiles[blankPos] = tiles[clickPos]
                    tiles[clickPos] = 0
                    blankPos = clickPos
                }
                repaint()
            }
        })

        shuffle()
    }

    private fun shuffle() {
        do {
            reset()
            // don't include the blank space in the shuffle,
            // leave it in the home position
            var n = 15
            while (n > 1) {
                val r = rand.nextInt(n--)
```

```
                val tmp = tiles[r]
                tiles[r] = tiles[n]
                tiles[n] = tmp
            }
        } while (!isSolvable())
    }

    private fun reset() {
        for (i in 0 until tiles.size) {
            tiles[i] = (i + 1) % tiles.size
        }
        blankPos = 15
    }

    /*  Only half the permutations of the puzzle are solvable.

        Whenever a tile is preceded by a tile with higher value it counts
        as an inversion. In our case, with the blank space in the home
        position, the number of inversions must be even for the puzzle
        to be solvable.
    */

    private fun isSolvable(): Boolean {
        var countInversions = 0
        for (i in 0 until 15) {
            (0 until i)
                .filter { tiles[it] > tiles[i] }
                .forEach { countInversions++ }
        }
        return countInversions % 2 == 0
    }

    private fun drawGrid(g: Graphics2D) {
        for (i in 0 until tiles.size) {
            if (tiles[i] == 0) continue

            val r = i / 4
            val c = i % 4
            val x = margin + c * tileSize
            val y = margin + r * tileSize

            with(g) {
                color = foreground
                fillRoundRect(x, y, tileSize, tileSize, 25, 25)
                color = Color.black
                drawRoundRect(x, y, tileSize, tileSize, 25, 25)
                color = Color.white
            }
            drawCenteredString(g, tiles[i].toString(), x, y)
        }
    }

    private fun drawCenteredString(g: Graphics2D, s: String, x: Int, y: Int) {
        val fm = g.fontMetrics
        val asc = fm.ascent
        val des = fm.descent

        val xx = x + (tileSize - fm.stringWidth(s)) / 2
        val yy = y + (asc + (tileSize - (asc + des)) / 2)

        g.drawString(s, xx, yy)
    }

    override fun paintComponent(gg: Graphics) {
        super.paintComponent(gg)
        val g = gg as Graphics2D
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON)
        drawGrid(g)
    }
}

fun main(args: Array<String>) {
    SwingUtilities.invokeLater {
        val f = JFrame()
        with(f) {
            defaultCloseOperation = JFrame.EXIT_ON_CLOSE
            title = "Fifteen Puzzle"
```

```
            isResizable = false
            add(FifteenPuzzle(640, 80), BorderLayout.CENTER)
            pack()
            setLocationRelativeTo(null)
            isVisible = true
        }
    }
}
```

Liberty BASIC

Translation of: Commodore BASIC
Works with: Just BASIC

```
' 15-PUZZLE GAME
' ******************************
dim d(16)
dim ds$(16) ' Board pieces
dim sh(3)

call buildBoard introAndLevel()
call printPuzzle
do
  print "To move a piece, enter its number: "
  input x
  while isMoveValid(x, y, z) = 0
    print "Wrong move."
    call printPuzzle
    print "To move a piece, enter its number: "
    input x
  wend
  d(z) = x
  d(y) = 0
  call printPuzzle
loop until isPuzzleComplete()
print "YOU WON!"
end

sub printPuzzle
  for p = 1 to 16
    if d(p) = 0 then
      ds$(p) = "     "
    else
      ds$(p) = using("###", d(p)) + "  "
    end if
  next p
  print "+-----+-----+-----+-----+"
  print "|"; ds$(1); "|"; ds$(2); "|"; ds$(3); "|"; ds$(4); "|"
  print "+-----+-----+-----+-----+"
  print "|"; ds$(5); "|"; ds$(6); "|"; ds$(7); "|"; ds$(8); "|"
  print "+-----+-----+-----+-----+"
  print "|"; ds$(9); "|"; ds$(10); "|";ds$(11); "|"; ds$(12); "|"
  print "+-----+-----+-----+-----+"
  print "|"; ds$(13); "|"; ds$(14); "|"; ds$(15); "|"; ds$(16); "|"
  print "+-----+-----+-----+-----+"
end sub

function introAndLevel()
  cls
  sh(1) = 10
  sh(2) = 50
  sh(3) = 100
  print "15 PUZZLE GAME"
  print
  print
  print "Please enter level of difficulty,"
  do
    print "1 (easy), 2 (medium) or 3 (hard): ";
    input level
  loop while (level < 1) or (level > 3)
  introAndLevel = level
end function

sub buildBoard level
```

```
  ' Set pieces in correct order first
  for p = 1 to 15
    d(p) = p
  next p
  d(16) = 0 ' 0 = empty piece/slot
  z = 16 ' z = empty position
  print
  print "Shuffling pieces";
  for n = 1 to sh(level)
    print ".";
    do
      x = int(rnd(0) * 4) + 1
      select case x
        case 1
          r = z - 4
        case 2
          r = z + 4
        case 3
          if (z - 1) mod 4 <> 0 then
            r = z - 1
          end if
        case 4
          if z mod 4 <> 0 then
            r = z + 1
          end if
      end select
    loop while (r < 1) or (r > 16)
    d(z) = d(r)
    z = r
    d(z) = 0
  next n
  cls
end sub

function isMoveValid(piece, byref piecePos, byref emptyPos)
  mv = 0
  if (piece >= 1) and (piece <= 15) then
    piecePos = piecePosition(piece)
    emptyPos = piecePosition(0)
    ' Check if empty piece is above, below, left or right to piece
    if (piecePos - 4 = emptyPos) or _
      (piecePos + 4 = emptyPos) or _
      ((piecePos - 1 = emptyPos) and (emptyPos mod 4 <> 0)) or _
      ((piecePos + 1 = emptyPos) and (piecePos mod 4 <> 0)) then
      mv = 1
    end if
  end if
  isMoveValid = mv
end function

function piecePosition(piece)
  p = 1
  while d(p) <> piece
    p = p + 1
    if p > 16 then
      print "UH OH!"
      stop
    end if
  wend
  piecePosition = p
end function

function isPuzzleComplete()
  pc = 0
  p = 1
  while (p < 16) and (d(p) = p)
    p = p + 1
  wend
  if p = 16 then
    pc = 1
  end if
  isPuzzleComplete = pc
end function
```

```
#Please note that all this code can be performed in livecode with just few mouse clicks
#This is just a pure script exampe
on OpenStack
   show me  #Usually not necessary
   #tile creation
   set the width of the templateButton to 50
   set the height of the templateButton to 50
   repeat with i=1 to 16
      create button
      set the label of  button i to i
      if i =1 then
         set the top of button 1 to 0
         set the left of button 1 to 0
      end if
      if i > 1 and 1 <=4  then
         set the left of button i to the right of button (i-1)
         set the top of button i to the top of button 1
      end if
      if i >= 5 and i <= 8 then
         set the top of button i to the bottom of button 1
         if i = 5 then
            set the left of button i to the left of button 1
         else
            set the left of button i to the right of button (i - 1)
         end if
      end if
      if i >= 9 and i <= 12 then
         set the top of button i to the bottom of button 5
         if i = 9 then
            set the left of button i to the left of button 1
         else
            set the left of button i to the right of button (i - 1)
         end if
      end if
      if i >= 13 and i <= 16 then
         set the top of button i to the bottom of button 9
         if i = 13 then
            set the left of button i to the left of button 1
         else
            set the left of button i to the right of button (i - 1)
         end if
      end if
      #this is usally the script directly wirtten in the objects, it's really weird this way
      put "on MouseUp" &CR& "if checkDistance(the label of me) then" & CR &"put the loc of me into temp" into ts
      put CR& "set the loc of me to the loc of button 16" after ts
      put CR& "set the loc of button 16 to temp" & Cr & "end if " &CR &"End MouseUp" after ts
      set the script of button i to ts
   end repeat
   #graphic adjustements
   set the visible of button 16 to false
   set the width of this stack to the right of button 16
   set the height of this stack to the bottom of button 16
end openStack

function checkDistance i
   if (((the top of button i - the bottom of button 16) = 0 OR (the top of button 16 - the bottom of button i) =
0) AND the left of button i = the left of button 16) OR (((the left of button i - the right of button 16) = 0 OR
(the right of button i - the left of button 16) = 0) AND the top of button i = the top of button 16) then
      return true
   else
      return false
   end if
end checkDistance
```

Screenshot: [2] (https://s24.postimg.org/uc6fx7kph/Livecode15_Puzzle_Game.png)

Lua

```lua
math.randomseed( os.time() )
local puz = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0 }
local dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } }
local sx, sy = 4, 4
```

```
function isValid( tx, ty )
    return tx > 0 and tx < 5 and ty > 0 and ty < 5
end
function display()
    io.write( "\n\n" )
    for j = 1, 4 do
        io.write( "+----+----+----+----+\n" )
        for i = 1, 4 do
            local d = puz[i + j * 4 - 4]
            io.write( ": " )
            if d < 10 then io.write( " " ) end
            if d < 1  then
                io.write( "   " )
            else
                io.write( string.format( "%d ", d ) )
            end
        end
        io.write( ":\n" )
    end
    io.write( "+----+----+----+----+\n\n" )
end
function getUserNove()
    local moves, r, tx, ty = {}
    for d = 1, 4 do
        tx = sx; ty = sy
        tx = tx + dir[d][1]; ty = ty + dir[d][2]

        if isValid( tx, ty ) then
            table.insert( moves, puz[tx + ty * 4 - 4] )
        end
    end

    io.write( "Your possible moves are: " )
    for i = 1, #moves do
        io.write( string.format( "%d ", moves[i] ) )
    end

    io.write ( "\nYour move: " ); r = tonumber( io.read() )
    if r ~= nil then
        for i = 1, #moves do
            if moves[i] == r then return r end
        end
    end
    print( "Invalid move!" )
    return -1
end
function checked( r )
    for i = 1, #puz do
        if puz[i] == r then
            puz[i] = 0
            sx = 1 + ( i - 1 ) % 4; sy = math.floor( ( i + 3 ) / 4 )
        elseif puz[i] == 0 then
            puz[i] = r
        end
    end
    for i = 2, #puz - 1 do
        if puz[i - 1] + 1 ~= puz[i] then return false end
    end
    return true
end
function beginGame()
    local r, d, tx, ty
    while( true ) do
        for i = 1, 100 do
            while( true ) do
                tx = sx; ty = sy; d = math.random( 4 )
                tx = tx + dir[d][1]; ty = ty + dir[d][2]
                if isValid( tx, ty ) then break end
            end
            puz[sx + sy * 4 - 4] = puz[tx + ty * 4 - 4]
            puz[tx + ty * 4 - 4] = 0; sx = tx; sy = ty
        end
        while( true ) do
            display(); r = getUserNove()
            if r > 0 then
                if checked( r ) then
                    display()
```

```
                    io.write( "\nDone!\n\nPlay again (Y/N)?" )
                    r = io.read()
                    if r ~= "Y" and r ~= "y" then
                        return
                    else
                        break
                    end
                end
            end
        end
    end
end
-- [ entry point ] --
beginGame()
```

Output:

```
+----+----+----+----+
:  1 : 15 :  7 :  3 :
+----+----+----+----+
: 14 :  9 :  2 :  4 :
+----+----+----+----+
:  5 : 13 :    : 10 :
+----+----+----+----+
:  6 : 12 :  8 : 11 :
+----+----+----+----+

Your possible moves are: 10 13 8 2
Your move:
```

Translation of: COMMODORE BASIC

M2000 Interpreter

Translation of: COMMODORE BASIC

I make a function RND() and I set Base for arrays to 1 Also I put some semi colons to Print statement Also I have to put variables after NEXT statements To be actual BASIC compatible we have to set FOR NEXT like BASIC, because for M2000 the direction defined by the starting-ending values, so always we get at least one time the block between FOR NEXT. So we can use:

SET SWITCHES "+FOR"

But here we don't have to skip FOR NEXT and no STEP clause used.

Also the code is not the best, because we can move from 4 position to 5 (we can't do that with real puzzle)

```
Module Puzzle15 {

      00 BASE 1 : DEF RND(X)=RND
      10 REM 15-PUZZLE GAME
      20 REM COMMODORE BASIC 2.0
      30 REM ******************************
      40 GOSUB 400 : REM INTRO AND LEVEL
      50 GOSUB 510 : REM SETUP BOARD
      60 GOSUB 210 : REM PRINT PUZZLE
      70 PRINT "TO MOVE A PIECE, ENTER ITS NUMBER:"
      80 INPUT X
      90 GOSUB 730 : REM CHECK IF MOVE IS VALID
      100 IF MV=0 THEN PRINT "WRONG MOVE" : GOSUB 1050 : GOTO 60
      110 D(Z)=X : D(Y)=0
      120 GOSUB 210 : REM PRINT PUZZLE
      130 GOSUB 950 : REM CHECK IF PUZZLE COMPLETE
      140 IF PC THEN 160
      150 GOTO 70
      160 PRINT"YOU WON!"
```

```
170 END
180 REM
190 REM ******************************
200 REM PRINT/DRAW THE PUZZLE
210 FOR P=1 TO 16
220   IF D(P)=0 THEN D$(P)="     " : GOTO 260
230   S$=STR$(D(P))
240   N=LEN(S$)
250   D$(P) = LEFT$("   ",3-N)+S$+"  "
260 NEXT P
270 PRINT "+-----+-----+-----+-----+"
280 PRINT "!";D$(1);"!";D$(2);"!";D$(3);"!";D$(4);"!"
290 PRINT "+-----+-----+-----+-----+"
300 PRINT "!";D$(5);"!";D$(6);"!";D$(7);"!";D$(8);"!"
310 PRINT "+-----+-----+-----+-----+"
320 PRINT "!";D$(9);"!";D$(10);"!";D$(11);"!";D$(12);"!"
330 PRINT "+-----+-----+-----+-----+"
340 PRINT "!";D$(13);"!";D$(14);"!";D$(15);"!";D$(16);"!"
350 PRINT "+-----+-----+-----+-----+"
360 RETURN
370 REM
380 REM ******************************
390 REM INTRO AND LEVEL OF DIFFICULTY
400 PRINT CHR$(147)
410 DIM SH(3) : SH(1)=10 : SH(2)=50 : SH(3)=100
420 PRINT "15 PUZZLE GAME FOR COMMODORE BASIC 2.0" : PRINT : PRINT
430 PRINT "PLEASE ENTER LEVEL OF DIFFICULTY,"
440 PRINT "1(EASY), 2(MEDIUM) OR 3(HARD):";
450 INPUT V
460 IF V<1 OR V>3 THEN 440
470 RETURN
480 REM
490 REM ******************************
500 REM BUILD THE BOARD
510 DIM D(16) : DIM D$(16) : REM BOARD PIECES
520 REM SET PIECES IN CORRECT ORDER FIRST
530 FOR P=1 TO 15
540   D(P) = P
550 NEXT P
560 D(16) = 0 : REM 0 = EMPTY PIECE/SLOT
570 Z=16      : REM Z = EMPTY POSITION
580 PRINT: PRINT "SHUFFLING PIECES";
590 FOR N=1 TO SH(V)
600   PRINT".";
610   X = INT(RND(0)*4)+1
620   R = Z+(X=1)*4-(X=2)*4+(X=3)-(X=4)
630   IF R<1 OR R>16 THEN 610
640   D(Z)=D(R)
650   Z=R
660   D(Z)=0
670 NEXT N
680 PRINT CHR$(147)
690 RETURN
700 REM
710 REM ******************************
720 REM CHECK IF MOVE IS VALID
730 MV = 0
740 IF X<1 OR X>15 THEN RETURN
750 REM FIND POSITION OF PIECE X
760 P=1
770 IF D(P)=X THEN Y=P : GOTO 810
780 P=P+1 : IF P>16 THEN PRINT "UH OH!" : STOP
790 GOTO 770
800 REM FIND POSITION OF EMPTY PIECE
810 P=1
820 IF D(P)=0 THEN Z=P : GOTO 860
830 P=P+1 : IF P>16 THEN PRINT "UH OH!" : STOP
840 GOTO 820
850 PRINT Y;Z
860 REM CHECK IF EMPTY PIECE IS ABOVE, BELOW, LEFT OR RIGHT TO PIECE X
870 IF Y-4=Z THEN MV=1 : RETURN
880 IF Y+4=Z THEN MV=1 : RETURN
890 IF Y-1=Z THEN MV=1 : RETURN
900 IF Y+1=Z THEN MV=1 : RETURN
910 RETURN
920 REM
930 REM ******************************
940 REM CHECK IF PUZZLE IS COMPLETE / GAME OVER
```

```
      950 PC = 0
      960 P=1
      970 IF D(P)<>P THEN RETURN
      980 P=P+1
      990 IF P<16 THEN 970
      1000 PC = 1
      1010 RETURN
      1020 REM
      1030 REM ****************************
      1040 REM A SMALL DELAY
      1050 FOR T=0 TO 400
      1060 NEXT T
      1070 RETURN
}
Puzzle15
```

Mathematica / Wolfram Language

```
grid = MapThread[{#1,#2} &, {Range @ 16, Range @ 16}]

Move[x_] := (empty = Select[grid, #[[1]]==16 &][[1,2]];
             If[(empty == x+4) || (empty == x-4) ||
                (Mod[empty,4] != 0 && empty == x-1) ||
                (Mod[empty,4] != 1 && empty == x+1),
             oldEmpty = grid[[empty]][[1]];
             grid[[empty]][[1]] = grid[[x]][[1]];
             grid[[x]][[1]] = oldEmpty])

CButton[{x_,loc_}] := If[x==16, Null, Button[x,Move @ loc]]

Dynamic @ Grid @ Partition[CButton /@ grid,4]
```

Mercury

The ideal in Mercury is to have a declarative module that encodes the game logic, and then separate modules to implement a human player with text commands (here), or keyed commands, or some kind of AI player, and so on. fifteen.print/3 is a arguably a smudge on fifteen's interface:

```
:- module fifteen.
:- interface.
:- use_module random, io.

:- type board.
:- type invalid_board
    --->    invalid_board(board).
:- type move
    --->    up
    ;       down
    ;       left
    ;       right.

    % init(Board):
    % Board is fifteen game in its initial state
    %
:- pred init(board::out) is det.

    % print(Board, !IO)
:- pred print(board::in, io.io::di, io.io::uo) is det.

    % Shuffled(Board, !RS):
    % Board is a fifteen game in a random (but valid) state.
    %
:- pred shuffled(board::out, random.supply::mdi, random.supply::muo) is det.

    % space(Board) = I:
    % I is the index of the blank space in the board.
    % Throws invalid_board iff there is no blank.
    %
:- func space(board) = int.
```

```
    % move(Move, !Board):
    % Move the blank space in a board in the given direction.
    % Fails if this is an invalid move to make.
    %
:- pred move(move::in, board::in, board::out) is semidet.

:- implementation.
:- import_module bt_array, int, list, string.
:- use_module array, exception.

:- type board == bt_array(int).

init(B) :- from_list(0, (1 .. 15) ++ [0], B).

print(B, !IO) :-
    Tile = (func(N) = ( if N = 0 then s("  ") else s(string.format("%2d", [i(N)])) )),
    io.format("\
|----+----+----+----|
| %s | %s | %s | %s |
| %s | %s | %s | %s |
| %s | %s | %s | %s |
| %s | %s | %s | %s |
|----+----+----+----|
",
        map(Tile, to_list(B)), !IO).

shuffled(!:B, !RS) :-
    init(!:B),
    some [!A] (
        array.from_list(to_list(!.B), !:A),
        array.random_permutation(!A, !RS),
        from_list(0, array.to_list(!.A), !:B)
    ).

space(Board) = I :- space(0, Board, I).

:- pred space(int::in, board::in, int::out) is det.
space(N, Board, I) :-
    ( if semidet_lookup(Board, N, X) then
        ( if X = 0 then
            N = I
        else
            space(N + 1, Board, I)
        )
    else
        exception.throw(invalid_board(Board))
    ).

:- pred swap(int::in, int::in, board::in, board::out) is det.
swap(I, J, !B) :-
    X = !.B ^ elem(I),
    Y = !.B ^ elem(J),
    !B ^ elem(I) := Y,
    !B ^ elem(J) := X.

move(M, !B) :- move(space(!.B), M, !B).

:- pred move(int::in, move::in, board::in, board::out) is semidet.
move(I, up, !B) :-
    I >= 4,
    swap(I, I - 4, !B).
move(I, down, !B) :-
    I < 12,
    swap(I, I + 4, !B).
move(I, left, !B) :-
    not (I = 0 ; I = 4 ; I = 8 ; I = 12),
    swap(I, I - 1, !B).
move(I, right, !B) :-
    not (I = 3 ; I = 7 ; I = 11 ; I = 15),
    swap(I, I + 1, !B).
```

As used:

```
:- module play_fifteen.
:- interface.
:- import_module io.
:- pred main(io::di, io::uo) is det.
:- implementation.
:- import_module string.
:- use_module random, fifteen, exception.

main(!IO) :-
    seed(Seed, !IO),
    random.init(Seed, RS),
    fifteen.shuffled(Board, RS, _),
    fifteen.print(Board, !IO),
    play(Board, !IO).

:- type input_command
    --->    up ; down ; left ; right
    ;       quit.

:- type command
    --->    move(fifteen.move)
    ;       quit.

:- pred input(input_command::in, command::out) is det.
input(up, move(fifteen.up)).
input(down, move(fifteen.down)).
input(left, move(fifteen.left)).
input(right, move(fifteen.right)).
input(quit, quit).

:- pred command(string::in, command::out) is semidet.
command(String, C) :-
    Term = chomp(String) ++ ".",
    io.read_from_string("play_fifteen", Term, length(Term),
        ok(I), io.posn(0, 0, 0), _),
    input(I, C).

:- pred play(fifteen.board::in, io::di, io::uo) is det.
play(!.B, !IO) :-
    io.write_string("Move? ", !IO),
    io.read_line_as_string(Res, !IO),
    (
        Res = eof
    ;
        Res = error(_),
        exception.throw(Res)
    ;
        Res = ok(String),
        ( if command(String, Command) then
            (
                Command = quit,
                io.write_string("Bye :(\n", !IO)
            ;
                Command = move(Move),
                ( if fifteen.move(Move, !B) then
                    ( if fifteen.init(!.B) then
                        fifteen.print(!.B, !IO),
                        io.write_string("You win!\n", !IO)
                    else
                        fifteen.print(!.B, !IO),
                        play(!.B, !IO)
                    )
                else
                    io.write_string("Invalid move.\n", !IO),
                    play(!.B, !IO)
                )
            )
        else
            io.write_string("I didn't understand that.\n", !IO),
            play(!.B, !IO)
        )
    ).

:- pragma foreign_decl("C", "#include <time.h>").
:- pred seed(int::out, io::di, io::uo) is det.
:- pragma foreign_proc("C",
```

```
    seed(Seed::out, _IO0::di, _IO::uo),
    [promise_pure, will_not_call_mercury],
"
    Seed = time(NULL);
").
```

MUMPS

```
15Game ;
        ; setting the layout
        f i=1:1:15 s box(i)=16-i
        ; no number for box 16
        s box(16)=""
        f  {
        ; initialise i for incrementation
                s i=0
        ; write out the 4-by-4 box
                f row=1:1:4 {
                        w !?20
                        f column=1:1:4 {
                                s i=$i(i)
                                w $j(box(i),2)," "
                        }
                }
                r !!,"Enter number to move (q to quit): ",number
                q:number="q"
                f i=1:1:16 q:box(i)=""
                if box(i)="" {
                        if i>4,number=box(i-4) {
                                s box(i)=number,box(i-4)=""
                                w !!
                        }
                        elseif i>1,i'=5,i'=9,i'=13,number=box(i-1) {
                                s box(i)=number,box(i-1)=""
                                w !!
                        }
                        elseif i<16,i'=4,i'=8,i'=12,number=box(i+1) {
                                s box(i)=number,box(i+1)=""
                                w !!
                        }
                        elseif i<13,number=box(i+4) {
                                s box(i)=number,box(i+4)=""
                                w !!
                        }
                        else {
                                w !,"You have to enter a number either above, below, left or right of the empty
space."
                        }
                }
        }
        q
```

Nim

```
import random, terminal

type
  Tile = uint8
  Board = array[16, Tile]

type
  Operation = enum
    opInvalid
    opUp
    opDown
    opLeft
    opRight
    opQuit
    opRestart

func charToOperation(c: char): Operation =
```

```
  case c
  of 'w', 'W': opUp
  of 'a', 'A': opLeft
  of 's', 'S': opDown
  of 'd', 'D': opRight
  of 'q', 'Q': opQuit
  of 'r', 'R': opRestart
  else: opInvalid

proc getKey(): Operation =
  var c = getch()
  if c == '\e':
    c = getch()
    if c == '[':
      case getch()
      of 'A': opUp
      of 'D': opLeft
      of 'B': opDown
      of 'C': opRight
      else: opInvalid
    else: charToOperation c
  else: charToOperation c

func isSolved(board: Board): bool =
  for i in 0..<board.high:
    if i != board[i].int - 1:
      return false
  true

func findTile(b: Board, n: Tile): int =
  for i in 0..b.high:
    if b[i] == n:
      return i

func canSwap(a, b: int): bool =
  let dist = a - b
  dist == 4 or dist == -4 or
    (dist == 1 and a mod 4 != 0) or
    (dist == -1 and b mod 4 != 0)

func pad(i: Tile): string =
  if i == 0:
    "│  "
  elif i < 10:
    "│ " & $i
  else:
    "│" & $i

proc draw(b: Board) =
  echo "┌──┬──┬──┬──┐\n",
     b[0].pad, b[1].pad, b[2].pad, b[3].pad,
     "│\n├──┼──┼──┼──┤\n",
     b[4].pad, b[5].pad, b[6].pad, b[7].pad,
     "│\n├──┼──┼──┼──┤\n",
     b[8].pad, b[9].pad, b[10].pad, b[11].pad,
     "│\n├──┼──┼──┼──┤\n",
     b[12].pad, b[13].pad, b[14].pad, b[15].pad,
     "│\n└──┴──┴──┴──┘"

proc clearScreen ()=
  for i in 1..10:
    eraseLine()
    cursorUp()

func calcPosMove(b: var Board; o: Operation; ): int =
  var posEmpty = b.findTile 0
  case o
  of opUp:    return posEmpty + 4
  of opDown:  return posEmpty - 4
  of opLeft:  return posEmpty + 1
  of opRight: return posEmpty - 1
  else: return -1

proc moveTile (b : var Board, op : Operation) : bool =
  let posMove = b.calcPosMove op
  let posEmpty = b.findTile 0
  if posMove < 16 and posMove >= 0 and canSwap(posEmpty, posMove):
    swap b[posEmpty], b[posMove]
```

```
    return true
  return false

proc shuffleBoard ( b: var Board, nSteps : int = 2000 ) =
  var opMove = @[opUp, opLeft, opDown, opRight]
  for i in 0 ..< nSteps:
    let op = opMove[rand(3)]
    discard b.moveTile op

proc generateBoard: Board =
  for i in 0..15:
    if i == 15 :
      result[i] = 0
    else:
      result[i] = (i + 1).Tile
  shuffleBoard result

when isMainModule:
  randomize()
  var
    board = generateBoard()
    empty = board.findTile 0

  block gameLoop:
    while not isSolved board:
      # draw
      draw board
      echo "Press arrow keys or WASD to move, R to Restart, Q to Quit"

      # handle input
      while true:
        let op = getKey()
        case op
        of opRestart:
          board = generateBoard()
          empty = board.findTile 0
          break
        of opQuit: break gameLoop
        of opInvalid: continue
        else:
          if board.moveTile op:
            empty = board.findTile 0
            break

      clearScreen()

    draw board
    echo "You win!"
```

Output:

```
+--+--+--+--+
|12|11|15| 2|
+--+--+--+--+
| 6|14| 3| 7|
+--+--+--+--+
| 5|  | 9|10|
+--+--+--+--+
| 1| 4| 8|13|
+--+--+--+--+
Press arrow keys or WASD to move, R to Restart, Q to Quit
```

OCaml

```
module Puzzle =
struct
  type t = int array
  let make () =
    [| 15; (* 0: the empty space *)
       0;  1;  2;  3;
       4;  5;  6;  7;
       8;  9; 10; 11;
```

```
      12; 13; 14;  |]

  let move p n =
    let hole, i = p.(0), p.(n) in
    p.(0) <- i;
    p.(n) <- hole

  let print p =
    let out = Array.make 16 "    " in
    for i = 1 to 15 do
      out.(p.(i)) <- Printf.sprintf " %2d" i
    done;
    for i = 0 to 15 do
      if (i mod 4) = 0 then print_newline ();
      print_string out.(i);
    done

  let shuffle p n =
    for i = 1 to n do
      move p (1 + Random.int 15)
    done
end

let play () =
  let p = Puzzle.make () in
  Puzzle.shuffle p 20;
  while true do
    Puzzle.print p;
    print_string " > ";
    Puzzle.move p (read_line () |> int_of_string)
  done
```

To move, input the number to slide into the blank. If you accidentally make an impossible move you can undo it by repeating the last input. A nice self-checked puzzle, the same as if you were physically moving the pieces around.

Output:

```
# play ();;

  8 11  7 13
  3  6    15
  9 10 12  4
  1  5 14  2 > 6

  8 11  7 13
  3     6 15
  9 10 12  4
  1  5 14  2 > 11

  8     7 13
  3 11  6 15
  9 10 12  4
  1  5 14  2 > 8

     8  7 13
  3 11  6 15
  9 10 12  4
  1  5 14  2 > 3

  3  8  7 13
    11  6 15
  9 10 12  4
  1  5 14  2 > 11

  3  8  7 13
 11     6 15
  9 10 12  4
  1  5 14  2 > 8

  3     7 13
 11  8  6 15
  9 10 12  4
  1  5 14  2 > 3
```

```
    3  7 13
 11  8  6 15
  9 10 12  4
  1  5 14  2 >
```

Pascal

This is Free Pascal(version >= 3.0.4) text mode implementation. To make a move, the user needs to enter the number of the selected tile.

```
program fifteen;
{$mode objfpc}
{$modeswitch advancedrecords}
{$coperators on}
uses
  SysUtils, crt;
type
  TPuzzle = record
  private
  const
    ROW_COUNT  = 4;
    COL_COUNT  = 4;
    CELL_COUNT = ROW_COUNT * COL_COUNT;
    RAND_RANGE = 101;
  type
    TTile          = 0..Pred(CELL_COUNT);
    TAdjacentCell  = (acLeft, acTop, acRight, acBottom);
    TPossibleMoves = set of TTile;
    TCellAdjacency = set of TAdjacentCell;
    TBoard         = array[0..Pred(CELL_COUNT)] of TTile;
  class var
    HBar: string;
  var
    FBoard: TBoard;
    FZeroPos,
    FMoveCount: Integer;
    FZeroAdjacency: TCellAdjacency;
    FPossibleMoves: TPossibleMoves;
    FSolved: Boolean;
    procedure DoMove(aTile: TTile);
    procedure CheckPossibleMoves;
    procedure PrintBoard;
    procedure PrintPossibleMoves;
    procedure TestSolved;
    procedure GenerateBoard;
    class constructor Init;
  public
    procedure New;
    function  UserMoved: Boolean;
    property  MoveCount: Integer read FMoveCount;
    property  Solved: Boolean read FSolved;
  end;

procedure TPuzzle.DoMove(aTile: TTile);
var
  Pos: Integer = -1;
  Adj: TAdjacentCell;
begin
  for Adj in FZeroAdjacency do
    begin
      case Adj of
        acLeft:   Pos := Pred(FZeroPos);
        acTop:    Pos := FZeroPos - COL_COUNT;
        acRight:  Pos := Succ(FZeroPos);
        acBottom: Pos := FZeroPos + COL_COUNT;
      end;
      if FBoard[Pos] = aTile then
        break;
    end;
  FBoard[FZeroPos] := aTile;
  FZeroPos := Pos;
  FBoard[Pos] := 0;
```

```
end;

procedure TPuzzle.CheckPossibleMoves;
var
  Row, Col: Integer;
begin
  Row := FZeroPos div COL_COUNT;
  Col := FZeroPos mod COL_COUNT;
  FPossibleMoves := [];
  FZeroAdjacency := [];
  if Row > 0 then
    begin
      FPossibleMoves += [FBoard[FZeroPos - COL_COUNT]];
      FZeroAdjacency += [acTop];
    end;
  if Row < Pred(ROW_COUNT) then
    begin
      FPossibleMoves += [FBoard[FZeroPos + COL_COUNT]];
      FZeroAdjacency += [acBottom];
    end;
  if Col > 0 then
    begin
      FPossibleMoves += [FBoard[Pred(FZeroPos)]];
      FZeroAdjacency += [acLeft];
    end;
  if Col < Pred(COL_COUNT) then
    begin
      FPossibleMoves += [FBoard[Succ(FZeroPos)]];
      FZeroAdjacency += [acRight];
    end;
end;

procedure TPuzzle.PrintBoard;
const
  Space = ' ';
  VBar  = '|';
  VBar1 = '| ';
  VBar2 = '|  ';
  VBar3 = '|    ';
var
  I, J, Pos, Tile: Integer;
  Row: string;
begin
  ClrScr;
  Pos := 0;
  WriteLn(HBar);
  for I := 1 to ROW_COUNT do
    begin
      Row := '';
      for J := 1 to COL_COUNT do
        begin
          Tile := Integer(FBoard[Pos]);
          case Tile of
            0:    Row += VBar3;
            1..9: Row += VBar2 + Tile.ToString + Space;
          else
            Row += VBar1 + Tile.ToString + Space;
          end;
          Inc(Pos);
        end;
      WriteLn(Row + VBar);
      WriteLn(HBar);
    end;
  if not Solved then
    PrintPossibleMoves;
end;

procedure TPuzzle.PrintPossibleMoves;
var
  pm: TTile;
  spm: string = '';
begin
  for pm in FPossibleMoves do
    spm += Integer(pm).ToString + ' ';
  WriteLn('possible moves: ', spm);
end;

procedure TPuzzle.TestSolved;
```

```
  function IsSolved: Boolean;
  var
    I: Integer;
  begin
    for I := 0 to CELL_COUNT - 3 do
      if FBoard[I] <> Pred(FBoard[Succ(I)]) then
        exit(False);
    Result := True;
  end;
begin
  FSolved := IsSolved;
  if not Solved then
    CheckPossibleMoves;
end;

procedure TPuzzle.GenerateBoard;
var
  I, CurrMove, SelMove: Integer;
  Tile: TTile;
begin
  FZeroPos := Pred(CELL_COUNT);
  FBoard[FZeroPos] := 0;
  for I := 0 to CELL_COUNT - 2 do
    FBoard[I] := Succ(I);
  for I := 1 to Random(RAND_RANGE) do
    begin
      CheckPossibleMoves;
      SelMove := 0;
      for Tile in FPossibleMoves do
        Inc(SelMove);
      SelMove := Random(SelMove);
      CurrMove := 0;
      for Tile in FPossibleMoves do
        begin
          if CurrMove = SelMove then
            begin
              DoMove(Tile);
              break;
            end;
          Inc(CurrMove);
        end;
    end;
end;

class constructor TPuzzle.Init;
var
  I: Integer;
begin
  HBar := '';
  for I := 1 to COL_COUNT do
    HBar += '+----';
  HBar += '+';
end;

procedure TPuzzle.New;
begin
  FSolved := False;
  FMoveCount := 0;
  GenerateBoard;
  CheckPossibleMoves;
  PrintBoard;
end;

function TPuzzle.UserMoved: Boolean;
const
  Sorry          = 'sorry, ';
  InvalidInput   = ' is invalid input';
  ImpossibleMove = ' is impossible move';
var
  UserInput: string;
  Tile: Integer = 0;
begin
  ReadLn(UserInput);
  case LowerCase(UserInput) of
    'c', 'cancel': exit(False);
  end;
  Result := True;
  if not Tile.TryParse(UserInput, Tile) then
```

```
    begin
      WriteLn(Sorry, UserInput, InvalidInput);
      exit;
    end;
  if not (Tile in [1..Pred(CELL_COUNT)]) then
    begin
      WriteLn(Sorry, Tile, InvalidInput);
      exit;
    end;
  if not (Tile in FPossibleMoves) then
    begin
      WriteLn(Sorry, Tile, ImpossibleMove);
      PrintPossibleMoves;
      exit;
    end;
  DoMove(Tile);
  Inc(FMoveCount);
  TestSolved;
  PrintBoard;
end;

procedure PrintStart;
begin
  ClrScr;
  WriteLn('Fifteen puzzle start:');
  WriteLn('  enter a tile number and press <enter> to move' );
  WriteLn('  enter Cancel(C) and press <enter> to exit' );
  Window(10, 4, 58, 21);
end;

procedure Terminate;
begin
  ClrScr;
  Window(1, 1, 80, 25);
  ClrScr;
  WriteLn('Fifteen puzzle exit.');
  Halt;
end;

function UserWantContinue(aMoveCount: Integer): Boolean;
var
  UserInput: string;
begin
  WriteLn('Congratulations! Puzzle solved in ', aMoveCount, ' moves.');
  WriteLn('Play again(Yes(Y)/<any button>)?');
  ReadLn(UserInput);
  case LowerCase(UserInput) of
    'y', 'yes': exit(True);
  end;
  Result := False;
end;

procedure Run;
var
  Puzzle: TPuzzle;
begin
  Randomize;
  PrintStart;
  repeat
    Puzzle.New;
    while not Puzzle.Solved do
      if not Puzzle.UserMoved then
        Terminate;
    if not UserWantContinue(Puzzle.MoveCount) then
      Terminate;
  until False;
end;

begin
  Run;
end.
```

Perl

Tk version

Library: Tk
Library: Perl/Tk

This Tk 15 puzzle implementation also shows the solvability of the current puzzle and the relative difficulty of it. On verbosity shows how the solvability is calculated. The program has some extra feature like font size and color scheme but also the possibility to set the intial board disposition. This program was originally posted by me at perlmonks (http://www.perlmonks.org/?node_id=1192660)

```
use strict;
use warnings;

use Getopt::Long;
use List::Util 1.29 qw(shuffle pairmap first all);
use Tk;
    # 5 options                                  1 label text
my ($verbose,@fixed,$nocolor,$charsize,$extreme,$solvability);

unless (GetOptions (
                     'verbose!' => \$verbose,
                     'tiles|positions=i{16}' => \@fixed,
                     'nocolor' => \$nocolor,
                     'charsize|size|c|s=i' => \$charsize,
                     'extreme|x|perl' => \$extreme,
                    )
        ) { die "invalid arguments!";}

@fixed = &check_req_pos(@fixed) if @fixed;

my $mw = Tk::MainWindow->new(-bg=>'black',-title=>'Giuoco del 15');

if ($nocolor){ $mw->optionAdd( '*Button.background',   'ivory' );}

$mw->optionAdd('*Button.font', 'Courier '.($charsize or 16).' bold' );
$mw->bind('<Control-s>', sub{#&init_board;
                             &shuffle_board});

my $top_frame = $mw->Frame( -borderwidth => 2, -relief => 'groove',
                          )->pack(-expand => 1, -fill => 'both');

$top_frame->Label( -textvariable=>\$solvability,
                  )->pack(-expand => 1, -fill => 'both');

my $game_frame = $mw->Frame(  -background=>'saddlebrown',
                              -borderwidth => 10, -relief => 'groove',
                            )->pack(-expand => 1, -fill => 'both');

# set victory conditions in pairs of coordinates
my @vic_cond =  pairmap {
       [$a,$b]
    } qw(0 0 0 1 0 2 0 3
         1 0 1 1 1 2 1 3
         2 0 2 1 2 2 2 3
         3 0 3 1 3 2 3 3);

my $board = [];

my $victorious = 0;

&init_board;

if ( $extreme ){ &extreme_perl}

&shuffle_board;

MainLoop;

##############################################################################
sub init_board{
  # tiles from 1 to 15
  for (0..14){
     $$board[$_]={
```

```
            btn=>$game_frame->Button(
                                -text => $_+1,
                                -relief => 'raised',
                                -borderwidth => 3,
                                -height => 2,
                                -width =>  4,
                                      -background=>$nocolor?'ivory':'gold1',
                                      -activebackground => $nocolor?'ivory':'gold1',
                                      -foreground=> $nocolor?'black':'DarkRed',
                                      -activeforeground=>$nocolor?'black':'DarkRed'
            ),
            name => $_+1,     # x and y set by shuffle_board
        };
        if (($_+1) =~ /^(2|4|5|7|10|12|13|15)$/ and !$nocolor){
            $$board[$_]{btn}->configure(
                                      -background=>'DarkRed',
                                      -activebackground => 'DarkRed',
                                      -foreground=> 'gold1',
                                      -activeforeground=>'gold1'
            );
        }
    }
    # empty tile
    $$board[15]={
            btn=>$game_frame->Button(
                                -relief => 'sunken',
                                -borderwidth => 3,
                                -background => 'lavender',
                                -height => 2,
                                -width =>  4,
            ),
            name => 16,      # x and y set by shuffle_board
        };
}
################################################################################
sub shuffle_board{
    if ($victorious){
        $victorious=0;
        &init_board;
    }
    if (@fixed){
          my $index = 0;

          foreach my $tile(@$board[@fixed]){
                  my $xy = $vic_cond[$index];
                  ($$tile{x},$$tile{y}) = @$xy;
                  $$tile{btn}->grid(-row=>$$xy[0], -column=> $$xy[1]);
                  $$tile{btn}->configure(-command =>[\&move,$$xy[0],$$xy[1]]);
                  $index++;
          }
          undef @fixed;
    }
    else{
        my @valid = shuffle (0..15);
        foreach my $tile ( @$board ){
            my $xy = $vic_cond[shift @valid];
            ($$tile{x},$$tile{y}) = @$xy;
            $$tile{btn}->grid(-row=>$$xy[0], -column=> $$xy[1]);
            $$tile{btn}->configure(-command => [ \&move, $$xy[0], $$xy[1] ]);
        }
    }
    my @appear =  map {$_->{name}==16?'X':$_->{name}}
                  sort{$$a{x}<=>$$b{x}||$$a{y}<=>$$b{y}}@$board;
    print "\n".('-' x 57)."\n".
          "Appearence of the board:\n[@appear]\n".
          ('-' x 57)."\n".
          "current\tfollowers\t               less than current\n".
          ('-' x 57)."\n" if $verbose;
    # remove the, from now on inutile, 'X' for the empty space
    @appear = grep{$_ ne 'X'} @appear;
    my $permutation;
    foreach my $num (0..$#appear){
        last if $num == $#appear;
         my $perm;
          $perm += grep {$_ < $appear[$num]} @appear[$num+1..$#appear];
          if ($verbose){
            print "[$appear[$num]]\t@appear[$num+1..$#appear]".
            (" " x (37 - length "@appear[$num+1..$#appear]")).
```

```
            "\t   $perm ".($num == $#appear  - 1 ? '=' : '+')."\n";
          }
          $permutation+=$perm;
    }
    print +(' ' x 50)."----\n" if $verbose;
    if ($permutation % 2){
        print "Impossible game with odd permutations!".(' ' x 13).
              "$permutation\n"if $verbose;
        $solvability = "Impossible game with odd permutations [$permutation]\n".
                       "(ctrl-s to shuffle)".
                       (($verbose or $extreme) ? '' :
                          " run with --verbose to see more info");
        return;
    }
    # 105 is the max permutation
    my $diff =  $permutation == 0 ? 'SOLVED' :
                $permutation < 35 ? 'EASY  ' :
                $permutation < 70 ? 'MEDIUM' : 'HARD  ';
    print "$diff game with even permutations".(' ' x 17).
          "$permutation\n" if $verbose;
    $solvability = "$diff game with permutation parity of [$permutation]\n".
                   "(ctrl-s to shuffle)";
}
###############################################################################
sub move{
    # original x and y
    my ($ox, $oy) = @_;
    my $self = first{$_->{x} == $ox and $_->{y} == $oy} @$board;
    return if $$self{name}==16;
    # check if one in n,s,e,o is the empty one
    my $empty = first {$_->{name} == 16 and
                          ( ($_->{x}==$ox-1 and $_->{y}==$oy) or
                            ($_->{x}==$ox+1 and $_->{y}==$oy) or
                            ($_->{x}==$ox and $_->{y}==$oy-1) or
                            ($_->{x}==$ox and $_->{y}==$oy+1)
                           )
                      } @$board;
    return unless $empty;
    # empty x and y
    my ($ex,$ey) = ($$empty{x},$$empty{y});
    # reconfigure emtpy tile
    $$empty{btn}->grid(-row => $ox, -column => $oy);
    $$empty{x}=$ox;    $$empty{y}=$oy;
    # reconfigure pressed tile
    $$self{btn}->grid(-row => $ex, -column => $ey);
    $$self{btn}->configure(-command => [ \&move, $ex, $ey ]);
    $$self{x}=$ex;    $$self{y}=$ey;
    # check for victory if the empty one is at the bottom rigth tile (3,3)
    &check_win if $$empty{x} == 3 and $$empty{y} == 3;
}
###############################################################################
sub check_win{
     foreach my $pos (0..$#$board){
        return unless ( $$board[$pos]->{'x'} == $vic_cond[$pos]->[0] and
                        $$board[$pos]->{'y'} == $vic_cond[$pos]->[1]);
     }
     # victory!
     $victorious = 1;
     my @text =  ('Dis','ci','pu','lus','15th','','','at',
                  'P','e','r','l','M','o','n','ks*');
     foreach my $tile(@$board){
            $$tile{btn}->configure( -text=> shift @text,
                                    -command=>sub{return});
            $mw->update;
            sleep 1;
     }
}
###############################################################################
sub check_req_pos{
    my @wanted = @_;
    # fix @wanted: seems GetOptions does not die if more elements are passed
    @wanted = @wanted[0..15];
    my @check = (1..16);
    unless ( all {$_ == shift @check} sort {$a<=>$b} @wanted ){
        die "tiles must be from 1 to 16 (empty tile)\nyou passed [@wanted]\n";
    }
    return map {$_-1} @wanted;
}
```

```
##############################################################################
sub extreme_perl {
  $verbose = 0;
  $mw->optionAdd('*font', 'Courier 20 bold');
  my @extreme = (
    'if $0',                                #1
    "\$_=\n()=\n\"foo\"=~/o/g",             #2
    "use warnings;\n\$^W ?\nint((length\n'Discipulus')/3)\n:'15'",   #3
    "length \$1\nif \$^X=~\n\/(?:\\W)(\\w*)\n(?:\\.exe)\$\/", #4
    "use Config;\n\$Config{baserev}",                   #5.
    "(split '',\nvec('JAPH'\n,1,8))[0]",       #6
    "scalar map\n{ord(\$_)=~/1/g}\nqw(p e r l)", #7
    "\$_ = () =\n'J A P H'\n=~\/\\b\/g",   # 8
    "eval join '+',\nsplit '',\n(substr\n'12345',3,2)",  #9
    'printf \'%b\',2',                      #10
    "int(((1+sqrt(5))\n/ 2)** 7 /\nsqrt(5)+0.5)-2",    #11
    "split '',\nunpack('V',\n01234567))\n[6,4]",  # 12
    'J','A','P','H'                                 # 13..16
  );
  foreach (0..15){
      $$board[$_]{btn}->configure(-text=> $extreme[$_],
                                  -height => 8,
                                   -width =>  16, ) if $extreme[$_];

  }
  @fixed = qw(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15);
  $mw->after(5000,\&shuffle_board);#
}
```

console version

This short console program just poses solvable puzzles: it achieves this shuffling a solved board n times, where n defaults to 1000 but can be passed as first argument of the program in the command line. It was originally posted by me at perlmonks (http://www.perlmonks.org/?node_id=1192865) but here a little modification was inserted to prevent wrong numbers to make the board messy.

```
use strict;
use warnings;

use List::Util qw(shuffle first);
my @tbl = ([1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]);
my $e = [3,3];
for (1..$ARGV[0]||1000) {
  my $new = (shuffle &ad($e))[0];
  $tbl[$e->[0]][$e->[1]] = $tbl[$new->[0]][$new->[1]];
  $tbl[$new->[0]]->[$new->[1]] = 16;
  $e = [$new->[0],$new->[1]];
}
while(1){
  print +(join ' ',map{$_==16?'  ':sprintf '%02s',$_}@{$tbl[$_]}),"\n" for 0..3;
  my $m = <STDIN>;
  chomp $m;
  die "Enter a number to move!" unless $m;
  if ($m > 15){
    warn "$m not in the board! Enter a tile from 1 to 15\n";
    next;
  }
  my $tile=first{$tbl[$$_[0]]->[$$_[1]]==$m}map{[$_,0],[$_,1],[$_,2],[$_,3]}0..3;
  my $new=first{$tbl[$$_[0]]->[$$_[1]]==16}&ad(grep{$tbl[$$_[0]]->[$$_[1]]==$m}
          map {[$_,0],[$_,1],[$_,2],[$_,3]}0..3);
  if ($new){$tbl[$$new[0]][$$new[1]]=$m;$tbl[$$tile[0]][$$tile[1]]=16;}
  system ($^O eq 'MSWin32' ? 'cls' : 'clear');
}
sub ad{
      my $e = shift; grep {$_->[0]<4 && $_->[1]<4 && $_->[0]>-1 && $_->[1]>-1}
```

```
        [$$e[0]-1,$$e[1]],[$$e[0]+1,$$e[1]],[$$e[0],$$e[1]-1],[$$e[0],$$e[1]+1]
}
```

Phix

console only

Kept simple. Obviously, increase the 5 random moves for more of a challenge.

```
constant ESC=27, UP=328, LEFT=331, RIGHT=333, DOWN=336
sequence board = tagset(15)&0, solve = board
integer pos = 16

procedure print_board()
    for i=1 to length(board) do
        puts(1,iff(i=pos?"   ":sprintf("%3d",{board[i]})))
        if mod(i,4)=0 then puts(1,"\n") end if
    end for
    puts(1,"\n")
end procedure

procedure move(integer d)
    integer new_pos = pos+{+4,+1,-1,-4}[d]
    if new_pos>=1 and new_pos<=16
    and (mod(pos,4)=mod(new_pos,4) -- same col, or row:
     or floor((pos-1)/4)=floor((new_pos-1)/4)) then
        {board[pos],board[new_pos]} = {board[new_pos],0}
        pos = new_pos
    end if
end procedure

for i=1 to 5 do move(rand(4)) end for
while 1 do
    print_board()
    if board=solve then exit end if
    integer c = find(wait_key(),{ESC,UP,LEFT,RIGHT,DOWN})-1
    if c=0 then exit end if
    move(c)
end while
puts(1,"solved!\n")
```

Output:

```
  1  2  3  4
  5     6  8
  9 10  7 11
 13 14 15 12

  1  2  3  4
  5  6     8
  9 10  7 11
 13 14 15 12

  1  2  3  4
  5  6  7  8
  9 10    11
 13 14 15 12

  1  2  3  4
  5  6  7  8
  9 10 11
 13 14 15 12

  1  2  3  4
  5  6  7  8
  9 10 11 12
 13 14 15
```

```
solved!
```

GUI version

Library: Phix/pGUI
Library: Phix/online

You can run this online here (http://phix.x10.mx/p2js/15game.htm).

```
--
-- demo\rosetta\15_puzzle_game.exw
-- ==============================
--
with javascript_semantics
include pGUI.e

constant title = "15 puzzle",
         help_text = """
Put the tiles back into order 1..15 and the hole last.
Rightarrow moves the tile to the left of the hole (rightwards) into it.
Leftarrow moves the tile to the right of the hole (leftwards) into it.
Uparrow moves the tile below the hole (upwards) into it.
Downarrow moves the tile above the hole (downwards) into it.
Press 'N' to start a new game.
"""

function show_help()
    IupMessage(title,help_text,bWrap:=false)
    return IUP_IGNORE -- (don't open browser help!)
end function

Ihandle canvas, dialog
cdCanvas cddbuffer, cdcanvas

sequence solved = tagset(15)&0, board
integer pos

procedure move(integer d)
    if d=5 then -- new game
        board = deep_copy(solved)
        pos = 16
        for i=1 to 1000 do move(rand(4)) end for
    elsif d then
        integer new_pos = pos+{+4,+1,-1,-4}[d]
        if new_pos>=1 and new_pos<=16
        and (mod(pos,4)=mod(new_pos,4) -- same col, or row:
         or floor((pos-1)/4)=floor((new_pos-1)/4)) then
            board[pos] = board[new_pos]
            board[new_pos] = 0
            pos = new_pos
        end if
    end if
end procedure
move(5) -- new game

function redraw_cb(Ihandle /*ih*/)
    integer {dw,dh} = IupGetIntInt(canvas, "DRAWSIZE"),
            ox,oy,              -- top right coords
            os,ts,              -- overall and tile size
            ht                  -- half tile, for number positioning

    if dw>=dh then
        ox = floor((dw-dh)/2)
        oy = 0
        os = dh
    else
        ox = 0
        oy = floor((dh-dw)/2)
        os = dw
    end if
    ts = floor((os-10)/4-7)
    ht = floor(ts/2+5)-10
```

```
    cdCanvasActivate(cddbuffer)
    cdCanvasSetBackground(cddbuffer, #FAF8EF)
    cdCanvasClear(cddbuffer)
    cdCanvasSetForeground(cddbuffer, #BBADA0)
    cdCanvasRoundedBox(cddbuffer, ox+5, ox+os-5, oy+5, oy+os-5, 10, 10)

    integer tx = ox+15
    for y=1 to 4 do
        integer ty = oy+15
        for x=1 to 4 do
            integer bxy = board[(4-x)*4+y]
            cdCanvasSetForeground(cddbuffer, #EEE4DA)
            cdCanvasRoundedBox(cddbuffer, tx, tx+ts-10, ty, ty+ts-10, 5, 5)
            if bxy then
                cdCanvasSetForeground(cddbuffer, #776E65)
                cdCanvasFont(cddbuffer, "Calibri", CD_BOLD, 32)
                cdCanvasText(cddbuffer, tx+ht, ty+ht, sprint(bxy))
            end if
            ty += ts+5
        end for
        tx += ts+5
    end for
    cdCanvasFlush(cddbuffer)
    string fmt = iff(board==solved?"%s - solved":"%s")
    IupSetStrAttribute(dialog,"TITLE",fmt,{title})
    return IUP_DEFAULT
end function

function map_cb(Ihandle ih)
    cdcanvas = cdCreateCanvas(CD_IUP, ih)
    cddbuffer = cdCreateCanvas(CD_DBUFFER, cdcanvas)
    cdCanvasSetTextAlignment(cddbuffer, CD_CENTER)
    return IUP_DEFAULT
end function

function key_cb(Ihandle /*ih*/, atom c)
    if c=K_ESC then return IUP_CLOSE end if -- (standard practice for me)
    if c=K_F5 then return IUP_DEFAULT end if -- (let browser reload work)
    if c=K_F1 then return show_help() end if
    move(find(upper(c),{K_UP,K_LEFT,K_RIGHT,K_DOWN,'N'}))
    IupUpdate(canvas)
    return IUP_IGNORE
end function

IupOpen()

canvas = IupCanvas("RASTERSIZE=520x540")
IupSetCallbacks(canvas, {"MAP_CB", Icallback("map_cb"),
                         "ACTION", Icallback("redraw_cb")})

dialog = IupDialog(canvas,`TITLE="%s",MINSIZE=440x450`,{title})
IupSetCallback(dialog, "KEY_CB", Icallback("key_cb"));

IupShow(dialog)
IupSetAttribute(canvas, "RASTERSIZE", NULL)
IupSetAttributeHandle(NULL,"PARENTDIALOG",dialog)
if platform()!=JS then
    IupMainLoop()
    IupClose()
end if
```

PHP

OOP and MVC design pattern has been used to facilitate any improvements, e.g. tiles with graphics instead of just numbers.

```
<?php
// Puzzle 15 Game - Rosseta Code - PHP 7 as the server-side script language.

// This program DOES NOT use cookies.

session_start([
```

```
  "use_only_cookies" => 0,
  "use_cookies" => 0,
  "use_trans_sid" => 1,
]);

class Location
{
  protected $column, $row;

  function __construct($column, $row){
    $this->column = $column;
    $this->row = $row;
  }
  function create_neighbor($direction){
    $dx = 0; $dy = 0;
    switch ($direction){
      case 0: case 'left':  $dx = -1; break;
      case 1: case 'right': $dx = +1; break;
      case 2: case 'up':    $dy = -1; break;
      case 3: case 'down':  $dy = +1; break;
    }
    return new Location($this->column + $dx, $this->row + $dy);
  }
  function equals($that){
    return $this->column == $that->column && $this->row == $that->row;
  }
  function is_inside_rectangle($left, $top, $right, $bottom){
    return $left <= $this->column && $this->column <= $right
        && $top <= $this->row && $this->row <= $bottom;
  }
  function is_nearest_neighbor($that){
    $s = abs($this->column - $that->column) + abs($this->row - $that->row);
    return $s == 1;
  }
}

class Tile
{
  protected $index;
  protected $content;
  protected $target_location;
  protected $current_location;

  function __construct($index, $content, $row, $column){
    $this->index = $index;
    $this->content = $content;
    $this->target_location = new Location($row, $column);
    $this->current_location = $this->target_location;
  }
  function get_content(){
    return $this->content;
  }
  function get_index(){
    return $this->index;
  }
  function get_location(){
    return $this->current_location;
  }
  function is_completed(){
    return $this->current_location->equals($this->target_location);
  }
  function is_empty(){
    return $this->content == NULL;
  }
  function is_nearest_neighbor($that){
    $a = $this->current_location;
    $b = $that->current_location;
    return $a->is_nearest_neighbor($b);
  }
  function swap_locations($that){
    $a = $this->current_location;
    $b = $that->current_location;
    $this->current_location = $b;
    $that->current_location = $a;
  }
}

class Model
```

```
{
  protected $N;
  protected $M;
  protected $tiles;

  function __construct($N, $M){
    $this->N = $N;
    $this->M = $M;
    $this->tiles[0] = new Tile(0, NULL, $N, $M);
    for ($k = 1; $k < $N * $M; $k++ ){
      $i = 1 + intdiv($k - 1, $M);
      $j = 1 + ($k - 1) % $M;
      $this->tiles[$k] = new Tile($k, (string)$k, $i, $j);
    }
    $number_of_shuffles = 1000;
    $i = 0;
    while ($i < $number_of_shuffles)
      if ($this->move_in_direction(random_int(0, 3)))
        $i++;
  }
  function get_N(){
    return $this->N;
  }
  function get_M(){
    return $this->M;
  }
  function get_tile_by_index($index){
    return $this->tiles[$index];
  }
  function get_tile_at_location($location){
    foreach($this->tiles as $tile)
      if ($location->equals($tile->get_location()))
        return $tile;
    return NULL;
  }
  function is_completed(){
    foreach($this->tiles as $tile)
      if (!$tile->is_completed())
        return FALSE;
    return TRUE;
  }
  function move($tile){
    if ($tile != NULL)
      foreach($this->tiles as $target){
        if ($target->is_empty() && $target->is_nearest_neighbor($tile)){
          $tile->swap_locations($target);
          break;
        }
      }
  }
  function move_in_direction($direction){
    foreach($this->tiles as $tile)
      if ($tile->is_empty())
        break;
    $location = $tile->get_location()->create_neighbor($direction);
    if ($location->is_inside_rectangle(0, 0, $this->M, $this->N)){
      $tile = $this->get_tile_at_location($location);
      $this->move($tile);
      return TRUE;
    }
    return FALSE;
  }
}

class View
{
  protected $model;

  function __construct($model){
      $this->model = $model;
  }
  function show(){
    $N = $this->model->get_N();
    $M = $this->model->get_M();
    echo "<form>";
    for ($i = 1; $i <= $N; $i++){
      for ($j = 1; $j <= $M; $j++){
        $tile = $this->model->get_tile_at_location(new Location($i, $j));
```

```
        $content = $tile->get_content();
        if ($content != NULL)
          echo "<span class='puzzle'>"
          .    "<input type='submit' class='puzzle' name='index'"
          .    "value='$content'>"
          .    "</span>";
        else
          echo "<span class='puzzle'></span>";
      }
      echo "<br>";
    }
    echo "</form>";
    if ($this->model->is_completed()){
      echo "<p class='end-game'>";
      echo "You win!";
      echo "</p>";
    }
  }
}

class Controller
{
  protected $model;
  protected $view;

  function __construct($model, $view){
    $this->model = $model;
    $this->view = $view;
  }
  function run(){
    if (isset($_GET['index'])){
      $index = $_GET['index'];
      $this->model->move($this->model->get_tile_by_index($index));
    }
    $this->view->show();
  }
}
?>

<!DOCTYPE html>
<html lang="en"><meta charset="UTF-8">
<head>
  <title>15 puzzle game</title>
  <style>
    .puzzle{width: 4ch; display: inline-block; margin: 0; padding: 0.25ch;}
    span.puzzle{padding: 0.1ch;}
    .end-game{font-size: 400%; color: red;}
  </style>
</head>
<body>
  <p><?php
    if (!isset($_SESSION['model'])){
      $width = 4; $height = 4;
      $model = new Model($width, $height);
    }
    else
      $model = unserialize($_SESSION['model']);
    $view = new View($model);
    $controller = new Controller($model, $view);
    $controller->run();
    $_SESSION['model'] = serialize($model);
  ?></p>
</body>
</html>
```

Picat

```
 import util.

main =>
    Board = {{1,2,3,4},
             {5,6,7,8},
             {9,10,11,12},
             {13,14,15,0}},
```

```
    Goal = copy_term(Board),
    shuffle(Board),
    play(Board,Goal).

shuffle(Board) =>
    foreach (_ in 1..1000)
        R1 = random() mod 4 + 1,
        C1 = random() mod 4 + 1,
        R2 = random() mod 4 + 1,
        C2 = random() mod 4 + 1,
        T := Board[R1,C1],
        Board[R1,C1] := Board[R2,C2],
        Board[R2,C2] := T
    end.

play(Board,Goal) =>
    while (Board != Goal)
        print_board(Board),
        possible_moves(Board,R0,C0,Moves),
        printf("Possible moves are: %w, 0 to exit. Your move? =>  ", Moves),
        S = read_line().strip(),
        if S == "0" || S == "" then
            halt
        else
            move_hole(Board,R0,C0,to_int(S))
        end
    end,
    print_board(Board),
    println("Puzzle solved.").

print_board(Board) =>
    N = len(Board),
    print("+----+----+----+----+\n"),
    foreach (R in 1..N)
        print("|"),
        foreach (C in 1..N)
            printf("%4d|", Board[R,C])
        end,
        nl
    end,
    println("+----+----+----+----+").

possible_moves(Board,R0,C0,Moves) =>
    N = len(Board),
    between(1,N,R0),
    between(1,N,C0),
    Board[R0,C0] == 0, !,
    NeibsOfHole = [(R1,C1) : (R1,C1) in [(R0-1,C0), (R0+1,C0), (R0,C0-1), (R0,C0+1)], R1 >= 1, R1 =< N, C1 >= 1,
C1 =< N],
    Moves = sort([Board[R,C] : (R,C) in NeibsOfHole]).

move_hole(Board,R0,C0,S) =>
    N = len(Board),
    between(1,N,R),
    between(1,N,C),
    Board[R,C] == S, !,
    Board[R0,C0] := S,
    Board[R,C] := 0.
```

Powershell

Had a go at this in PowerShell. The board locks when you complete the puzzle. Board is always scrambled from a solved board so it's always solvable. Written in ISE with ISESteroids. I'm sure the code could be improved on.

```
#15 Puzzle Game
$Script:Neighbours = @{
    "1" = @("2","5")
    "2" = @("1","3","6")
    "3" = @("2","4","7")
    "4" = @("3","8")
    "5" = @("1","6","9")
```

```
    "6" = @("2","5","7","10")
    "7" = @("3","6","8","11")
    "8" = @("4","7","12")
    "9" = @("5","10","13")
    "10" = @("6","9","11","14")
    "11" = @("7","10","12","15")
    "12" = @("8","11","0")
    "13" = @("9","14")
    "14" = @("10","13","15")
    "15" = @("11","14","0")
    "0" = @("12","15")
}
$script:blank = ''
#region XAML window definition
$xaml = @'
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" mc:Ignorable="d"
   MinWidth="200"
   Width ="333.333"
   Title="15 Game"
   Topmost="True" Height="398.001" VerticalAlignment="Center" HorizontalAlignment="Center">
    <Grid HorizontalAlignment="Center" Height="285" Margin="0" VerticalAlignment="Center" Width="300">
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Button x:Name="B_1" Content="01" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24"/>
        <Button x:Name="B_2" Content="02" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="1"/>
        <Button x:Name="B_3" Content="03" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="2"/>
        <Button x:Name="B_4" Content="04" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="3"/>
        <Button x:Name="B_5" Content="05" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Row="1"/>
        <Button x:Name="B_6" Content="06" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="1" Grid.Row="1"/>
        <Button x:Name="B_7" Content="07" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="2" Grid.Row="1"/>
        <Button x:Name="B_8" Content="08" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="3" Grid.Row="1"/>
        <Button x:Name="B_9" Content="09" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Margin="0,71,0,0" Grid.Row="1" Grid.RowSpan="2"/>
        <Button x:Name="B_10" Content="10" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="1" Grid.Row="1" Margin="0,71,0,0" Grid.RowSpan="2"/>
        <Button x:Name="B_11" Content="11" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="2" Grid.Row="1" Margin="0,71,0,0" Grid.RowSpan="2"/>
        <Button x:Name="B_12" Content="12" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="3" Grid.Row="1" Margin="0,71,0,0" Grid.RowSpan="2"/>
        <Button x:Name="B_13" Content="13" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Margin="0,71,0,0" Grid.Row="2" Grid.RowSpan="2"/>
        <Button x:Name="B_14" Content="14" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="1" Grid.Row="2" Margin="0,71,0,0" Grid.RowSpan="2"/>
        <Button x:Name="B_15" Content="15" HorizontalAlignment="Center" VerticalAlignment="Center" Width="75"
Height="72" FontSize="24" Grid.Column="2" Grid.Row="2" Margin="0,71,0,0" Grid.RowSpan="2"/>
        <Button x:Name="B_0" Content="00" HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
Height="72" FontSize="24" Grid.Column="3" Grid.Row="2" Margin="0,71,0,0" Grid.RowSpan="2"/>
        <Button x:Name="B_Jumble" Grid.ColumnSpan="2" Content="Jumble Tiles" Grid.Column="1"
HorizontalAlignment="Left" Height="23" Margin="0,81,0,-33" Grid.Row="3" VerticalAlignment="Top" Width="150"/>
    </Grid>
</Window>
'@
#endregion

#region Code Behind
function Convert-XAMLtoWindow
```

```
{
    param
    (
        [Parameter(Mandatory=$true)]
        [string]
        $XAML
    )

    Add-Type -AssemblyName PresentationFramework

    $reader = [XML.XMLReader]::Create([IO.StringReader]$XAML)
    $result = [Windows.Markup.XAMLReader]::Load($reader)
    $reader.Close()
    $reader = [XML.XMLReader]::Create([IO.StringReader]$XAML)
    while ($reader.Read())
    {
        $name=$reader.GetAttribute('Name')
        if (!$name) { $name=$reader.GetAttribute('x:Name') }
        if($name)
        {$result | Add-Member NoteProperty -Name $name -Value $result.FindName($name) -Force}
    }
    $reader.Close()
    $result
}
function Show-WPFWindow
{
    param
    (
        [Parameter(Mandatory=$true)]
        [Windows.Window]
        $Window
    )

    $result = $null
    $null = $window.Dispatcher.InvokeAsync{
        $result = $window.ShowDialog()
        Set-Variable -Name result -Value $result -Scope 1
    }.Wait()
    $result
}
#endregion Code Behind

#region Convert XAML to Window
$window = Convert-XAMLtoWindow -XAML $xaml

#endregion

#region Define Event Handlers
function Test-Victory{
    #Evaluate if all the labels are in the correct position
    $victory = $true
    foreach($num in 1..15){
        if([int]$window."b_$num".Content -ne $num){$victory = $false;break}
    }
    return($victory)
}
function Test-Move($Number){
    #Number is a string of the pressed button number.
    if($Script:Neighbours[$Number] -contains $script:blank){
        return($true)
    } else {
        return($false)
    }
}
Function Move-Tile($Number,$Bypass){
    if((!(Test-Victory)) -or $Bypass){
        if(Test-Move $Number){
            #Set the new window label
            $window."B_$script:blank".content = $window."B_$Number".content
            $window."B_$script:blank".background = $window."B_$Number".background
            $window."B_$Number".background = "#FFDDDDDD"
            #Set the new blank window label
            $window."B_$Number".content = ''
            #Enable the old blank tile
            $window."B_$script:blank".isenabled = $true
            #disable the new blank tile
            $window."B_$Number".isenabled = $false
            #set the new blank
```

```
            $script:blank = $Number
        }
    }
}
function Move-TileRandom{
    $lastmove = "1"
    for($i=0;$i -lt 500;$i++){
        $move = $Script:Neighbours[$script:blank] | Where-Object {$_ -ne $lastmove} | Get-Random
        $lastmove = $move
        Move-Tile $move $true
    }
}
function Set-TileColour($Tile){
    #I was curious about setting tiles to a checkerboard pattern at this stage. It's probably far better to just
define it in the xaml
    #Ignore the blank tile
    if($Tile -ne 0){
        #check if the row of the tile is odd or even
        if((([math]::floor(($Tile - 1)/4)) % 2 -eq 0)){
            #check if the tile is odd or even
            if($Tile % 2 -eq 0){
                $window."B_$Tile".Background = "#FFFF7878"
            } else {
                $window."B_$Tile".Background = "#FF9696FF"
            }
        }else{
            if($Tile % 2 -eq 0){
                $window."B_$Tile".Background = "#FF9696FF"
            } else {
                $window."B_$Tile".Background = "#FFFF7878"
            }
        }
    } else {
        $window.B_0.Background = "#FFDDDDDD"
    }
}
$window.B_1.add_Click{
    $n = "1"
    move-tile $n
}
$window.B_2.add_Click{
    $n = "2"
    move-tile $n
}
$window.B_3.add_Click{
    $n = "3"
    move-tile $n
}
$window.B_4.add_Click{
    $n = "4"
    move-tile $n
}
$window.B_5.add_Click{
    $n = "5"
    move-tile $n
}
$window.B_6.add_Click{
    $n = "6"
    move-tile $n
}
$window.B_7.add_Click{
    $n = "7"
    move-tile $n
}
$window.B_8.add_Click{
    $n = "8"
    move-tile $n
}
$window.B_9.add_Click{
    $n = "9"
    move-tile $n
}
$window.B_10.add_Click{
    $n = "10"
    move-tile $n
}
$window.B_11.add_Click{
    $n = "11"
```

```
    move-tile $n
}
$window.B_12.add_Click{
    $n = "12"
    move-tile $n
}
$window.B_13.add_Click{
    $n = "13"
    move-tile $n
}
$window.B_14.add_Click{
    $n = "14"
    move-tile $n
}
$window.B_15.add_Click{
    $n = "15"
    move-tile $n
}
$window.B_0.add_Click{
    $n = "0"
    move-tile $n
}

$window.B_Jumble.add_Click{
    Move-TileRandom
}
#endregion Event Handlers
(([math]::floor(("9" - 1)/4)) % 2 -eq 0)
#region Manipulate Window Content
#initial processing of tiles
$array = 0..15 | ForEach-Object {"{0:00}" -f $_}
0..15 | ForEach-Object {if($array[$_] -ne '00'){$window."B_$_".content = $array[$_]} else {$window."B_$_".content
= '';$script:blank = "$_";$window."B_$_".isenabled = $false};Set-TileColour $_}
#Shove them around a bit
Move-TileRandom
#endregion
# Show Window
$result = Show-WPFWindow -Window $window
```

Processing

```
int number_of_grid_cells = 16;  //   Set the number of cells of the board here 9, 16, 25 etc
color piece_color = color(255, 175, 0);
color background_color = color(235, 231, 178);
color piece_shadow_dark = color(206, 141, 0);
color piece_shadow_light = color(255, 214, 126);
int z, t, p, piece_number, row_length, piece_side_length;

PuzzlePiece[] piece = new PuzzlePiece[number_of_grid_cells]; //  Number of puzzle pieces objects array

void setup() {
  size(400, 400);  // Window size width and height must be egual
    background(200, 50, 0);
  row_length = int(sqrt(number_of_grid_cells));
  piece_side_length = width/row_length;
  textSize(piece_side_length/2.7);
  textAlign(CENTER);

  PVector[] xy_values = new PVector[number_of_grid_cells]; //  Setting the x and y values for each cell on grid
  for (int i = 0; i < number_of_grid_cells; i += row_length) { // Values are the top left pixel of the cell
    for (int j = 0; j < row_length; j++) {
      xy_values[z] = new PVector();
      xy_values[z].x = j*piece_side_length;
      xy_values[z].y = t*piece_side_length;
      z++;
    }
    t++;
  }

  int[] place = new int[number_of_grid_cells]; // This array is  to help placing the pieces randomly and store
values in piece objects array
  for (int i = 0; i < number_of_grid_cells; i++) place[i] = 0;
  piece_number = 0;
```

```
  while (piece_number < number_of_grid_cells) {  // Placing pieces randomly in grid
    p = int(random(0, number_of_grid_cells));
    if (place[p] == 0) { // Once placed will be set to 1 to avoid designing again at this location
      piece[piece_number] = new PuzzlePiece(piece_number, xy_values[p].x, xy_values[p].y); // Creating the piece objects array
      place[p] = 1;
      piece[piece_number].design(); // Design newly create piece object
      piece_number++;
    }
  }
}

void draw() {
  for (int i = 0; i < number_of_grid_cells; i++) {   // Search all piece object indexes and verify which one is mouse pressed in this loop
    if (mousePressed == true && mouseX >= piece[i].xPosition() && mouseX <= piece[i].xPosition()+piece_side_length && mouseY >= piece[i].yPosition() && mouseY <= piece[i].yPosition()+piece_side_length && piece[i].pieceNumber() != 15) {
      if (pieceMove(piece[number_of_grid_cells-1].xPosition(), piece[number_of_grid_cells-1].yPosition(), piece[i].xPosition(), piece[i].yPosition())) {
        float temp_x = piece[number_of_grid_cells-1].xPosition(); // Remember x and y value of final piece index (white piece)
        float temp_y = piece[number_of_grid_cells-1].yPosition();
        piece[number_of_grid_cells-1].storePos(piece[i].xPosition(), piece[i].yPosition()); // Store clicked x and y value in final index of piece array
        piece[i].storePos(temp_x, temp_y); // Store temp x and y value (the last/previous final index values) in current clicked piece index
        piece[number_of_grid_cells-1].design(); // draw the final index piece index (only final piece index is painted white)
        piece[i].design(); // Draw a numbered piece of current index
      }
    }
  }
}

boolean pieceMove(float final_index_piece_x, float final_index_piece_y, float current_index_x, float current_index_y) {
  // If both x values from clicked and white piece have same value meaning in same horizontal column
  // AND   current clicked y value is equal to white piece y value - piece side lenght  OR  current clicked y value + piece side lenght is egual to white piece y
  if (current_index_x == final_index_piece_x && (current_index_y == final_index_piece_y-piece_side_length || (current_index_y == final_index_piece_y+piece_side_length))) return true;
  // If both y values from clicked and white piece have same value meaning in same vertical column
  // AND   current clicked x value is equal to white piece x value - piece side lenght  OR  current clicked x value + piece side lenght is egual to white piece x
  else if (current_index_y == final_index_piece_y && (current_index_x == final_index_piece_x-piece_side_length || (current_index_x == final_index_piece_x+piece_side_length))) return true;
  else return false;
}

class PuzzlePiece {
  int piece_number;
  float x_pos, y_pos;

  PuzzlePiece(int _piece_nr, float _xp, float _yp) {
    piece_number = _piece_nr;
    x_pos = _xp;
    y_pos = _yp;
  }

  void storePos(float _xp, float _yp) {
    x_pos = _xp;
    y_pos = _yp;
  }

  int pieceNumber() {
    return piece_number;
  }

  float xPosition() {
    return x_pos;
  }

  float yPosition() {
    return y_pos;
  }

  void design() {
```

```
    noStroke();
    fill(piece_color);
    if (piece_number == number_of_grid_cells-1) fill(background_color);
    rect(x_pos+1, y_pos+1, piece_side_length-1, piece_side_length-1);
    if (piece_number != number_of_grid_cells-1) {
      fill(0); // Black text shadow
      text(piece_number+1, x_pos+piece_side_length/2+1, y_pos+piece_side_length/2+textAscent()/2);
      fill(255);
      text(piece_number+1, x_pos+piece_side_length/2, y_pos+piece_side_length/2+textAscent()/2);
      stroke(piece_shadow_dark);
      line(x_pos+piece_side_length-1, y_pos+1, x_pos+piece_side_length-1, y_pos+piece_side_length-1); // Right
side shadow
      line(x_pos+2, y_pos+piece_side_length, x_pos+piece_side_length-1, y_pos+piece_side_length); // Bottom side
shadow
      stroke(piece_shadow_light);
      line(x_pos+2, y_pos-1, x_pos+2, y_pos+piece_side_length); // Left bright
      line(x_pos+2, y_pos+1, x_pos+piece_side_length-1, y_pos+1); // Upper bright
    }
  }
}
```

It can be played on line :
here. (https://www.openprocessing.org/sketch/863055/)

Processing Python mode

Translation of: Processing

```
# Set the number of cells of the board here 9, 16, 25 etc
num_grid_cells = 16
piece_color = color(255, 175, 0)
background_color = color(235, 231, 178)
piece_shadow_dark = color(206, 141, 0)
piece_shadow_light = color(255, 214, 126)

def setup():
    global piece, piece_number, row_length, piece_side_length
    size(400, 400)  # Window size width and height must be egual
    background(200, 50, 0)
    row_length = int(sqrt(num_grid_cells))
    piece_side_length = width / row_length
    textSize(piece_side_length / 2.7)
    textAlign(CENTER)
    # Setting the x and y values for each cell on grid
    xy_val = []
    t = 0
    for i in range(0, num_grid_cells, row_length):
        for j in range(row_length):
            xy_val.append((j * piece_side_length,
                           t * piece_side_length))
        t += 1
    piece = []  # Puzzle piece objects
    placed = [False] * num_grid_cells  # to help placing the pieces randomly
    piece_number = 0
    # Placing pieces randomly in grid
    while (piece_number < num_grid_cells):
        p = int(random(0, num_grid_cells))
        # Once placed will be set to True to avoid adding again at this location
        if not placed[p]:
            # Creating the piece objects list
            piece.append(PuzzlePiece(piece_number, xy_val[p][0], xy_val[p][1]))
            placed[p] = True
            piece[piece_number].design()  # Draw newly create piece object
            piece_number += 1

def draw():
    # Search all piece object indexes and verify which one is mouse pressed
    for i in range(num_grid_cells):
        if (mousePressed and
                piece[i].x <= mouseX <= piece[i].x + piece_side_length and
                piece[i].y <= mouseY <= piece[i].y + piece_side_length and
                piece[i].piece_number != 15):
            if (pieceMove(piece[num_grid_cells - 1].x, piece[num_grid_cells - 1].y, piece[i].x, piece[i].y)):
                # Remember x and y value of final piece index (white piece)
```

```
                temp_x = int(piece[num_grid_cells - 1].x)
                temp_y = int(piece[num_grid_cells - 1].y)
                # Store clicked x and y value in final index of piece list
                piece[num_grid_cells - 1].set_pos(piece[i].x, piece[i].y)
                # Store temp x and y value (the last/previous final index
                # values) in current clicked piece
                piece[i].set_pos(temp_x, temp_y)
                # draw the final index piece index (only final piece index is
                # painted white)
                piece[num_grid_cells - 1].design()
                piece[i].design()  # Draw a numbered piece of current index

def pieceMove(final_index_piece_x, final_index_piece_y, current_index_x, current_index_y):
    # If both x's from clicked and white piece have same value meaning in same horizontal column
    # AND current clicked y value is equal to white piece y value - piece side lenght OR
    # current clicked y value + piece side lenght is egual to white piece y
    if (current_index_x == final_index_piece_x and (current_index_y == final_index_piece_y - piece_side_length or
                                                    (current_index_y == final_index_piece_y +
piece_side_length))):
        return True
    # If both y's from clicked and white piece have same value meaning in same vertical column AND current clicked
x value
    # is equal to white piece x value - piece side lenght OR current clicked x value + piece side lenght is
    # egual to white piece x
    elif (current_index_y == final_index_piece_y and (current_index_x == final_index_piece_x - piece_side_length
or
                                                      (current_index_x == final_index_piece_x +
piece_side_length))):
        return True
    else:
        return False

class PuzzlePiece:

    def __init__(self, pn, xp, yp):
        self.piece_number = pn
        self.x = xp
        self.y = yp

    def set_pos(self, xp, yp):
        self.x = xp
        self.y = yp

    def design(self):
        noStroke()
        fill(piece_color)
        if (self.piece_number == num_grid_cells - 1):
            fill(background_color)
        rect(self.x + 1, self.y + 1,
             piece_side_length - 1, piece_side_length - 1)
        if (self.piece_number != num_grid_cells - 1):
            fill(0)  # Black text shadow
            text(self.piece_number + 1, self.x + piece_side_length / 2 + 2,
                 self.y + piece_side_length / 2 + textAscent() / 2)
            fill(255)
            text(self.piece_number + 1, self.x + piece_side_length / 2,
                 self.y + piece_side_length / 2 + textAscent() / 2)
            stroke(piece_shadow_dark)
            line(self.x + piece_side_length - 1, self.y + 1, self.x +
                 piece_side_length - 1, self.y + piece_side_length - 1)  # Right side shadow
            line(self.x + 2, self.y + piece_side_length, self.x +
                 piece_side_length - 1, self.y + piece_side_length)  # Bottom side shadow
            stroke(piece_shadow_light)
            # Left bright
            line(self.x + 2, self.y - 1, self.x + 2,
                 self.y + piece_side_length)
            # Upper bright
            line(self.x + 2, self.y + 1, self.x +
                 piece_side_length - 1, self.y + 1)
```

PureBasic

Console version. All Puzzles are solvable. #Difficulty is the number of shuffle (default 10). Numbers are displayed in Hexadecimal (ie 1 to F) Default controls are u,d,l,r

```
#difficulty=10 ;higher is harder
#up="u"
#down="d"
#left="l"
#right="r"
OpenConsole("15 game"):EnableGraphicalConsole(1)
Global Dim Game.i(3,3)
Global hole.point

Procedure NewBoard()
    num.i=0
    For j=0 To 3
        For i=0 To 3
            Game(i,j)=num
            num+1
        Next i
    Next j
EndProcedure
Procedure.s displayBoard()
    For j=0 To 3
        For i=0 To 3
            ConsoleLocate(i,j)
            If Game(i,j)=0:Print(" "):Continue:EndIf
            Print(Hex(Game(i,j)))
        Next i
    Next j
    PrintN("")
    Print("Your Choice Up :"+#up+", Down :"+#down+", Left :"+#left+" Or Right :"+#right+" ")
    Repeat
        keypress$=Inkey()
    Until keypress$<>""
    keypress$=LCase(keypress$)
    ProcedureReturn keypress$
EndProcedure
Procedure UpdateBoard(key$)
    If key$=#up And hole\y<3
        Swap game(hole\x,hole\y),game(hole\x,hole\y+1):hole\y+1
    ElseIf key$=#down And hole\y
        Swap game(hole\x,hole\y),game(hole\x,hole\y-1):hole\y-1
    ElseIf key$=#left And hole\x<3
        Swap game(hole\x,hole\y),game(hole\x+1,hole\y):hole\x+1
    ElseIf key$=#right And hole\x
        Swap game(hole\x,hole\y),game(hole\x-1,hole\y):hole\x-1
    EndIf
EndProcedure
Procedure TestGameWin()
    For j=0 To 3
        For i=0 To 3
            num+1
            If game(i,j)=num:win+1:EndIf
        Next i
    Next j
    If win=15:ProcedureReturn 1:EndIf
EndProcedure
Procedure ShuffleBoard(difficulty.i)
    Dim randomKey$(3)
    randomkey$(0)=#up:randomkey$(1)=#down:randomkey$(2)=#left:randomkey$(3)=#right
    For i=1 To difficulty
       UpdateBoard(randomKey$(Random(3)))
    Next i
EndProcedure
NewBoard()
ShuffleBoard(#difficulty)
Repeat
    choice$=displayBoard()
    UpdateBoard(choice$)
Until TestGameWin()
Print("Won !")
CloseConsole()
```

```
1234
5678
9ABC
DEF
```

```
Your Choice Up :u, Down :d, Left :l Or Right :r
```

Python

Python: Original, with output

Works with: Python version 3.X

unoptimized

```
''' Structural Game for 15 - Puzzle with different difficulty levels'''
from random import randint


class Puzzle:
    def __init__(self):
        self.items = {}
        self.position = None

    def main_frame(self):
        d = self.items
        print('+-----+-----+-----+-----+')
        print('|%s|%s|%s|%s|' % (d[1], d[2], d[3], d[4]))
        print('+-----+-----+-----+-----+')
        print('|%s|%s|%s|%s|' % (d[5], d[6], d[7], d[8]))
        print('+-----+-----+-----+-----+')
        print('|%s|%s|%s|%s|' % (d[9], d[10], d[11], d[12]))
        print('+-----+-----+-----+-----+')
        print('|%s|%s|%s|%s|' % (d[13], d[14], d[15], d[16]))
        print('+-----+-----+-----+-----+')

    def format(self, ch):
        ch = ch.strip()
        if len(ch) == 1:
            return '  ' + ch + '  '
        elif len(ch) == 2:
            return '  ' + ch + ' '
        elif len(ch) == 0:
            return '      '

    def change(self, to):
        fro = self.position
        for a, b in self.items.items():
            if b == self.format(str(to)):
                to = a
                break
        self.items[fro], self.items[to] = self.items[to], self.items[fro]
        self.position = to

    def build_board(self, difficulty):
        for i in range(1, 17):
            self.items[i] = self.format(str(i))
        tmp = 0
        for a, b in self.items.items():
            if b == '  16 ':
                self.items[a] = '      '
                tmp = a
                break
        self.position = tmp
        if difficulty == 0:
            diff = 10
        elif difficulty == 1:
            diff = 50
        else:
            diff = 100
        for _ in range(diff):
            lst = self.valid_moves()
            lst1 = []
            for j in lst:
                lst1.append(int(j.strip()))
            self.change(lst1[randint(0, len(lst1)-1)])
```

```
    def valid_moves(self):
        pos = self.position
        if pos in [6, 7, 10, 11]:
            return self.items[pos - 4], self.items[pos - 1],\
                   self.items[pos + 1], self.items[pos + 4]
        elif pos in [5, 9]:
            return self.items[pos - 4], self.items[pos + 4],\
                   self.items[pos + 1]
        elif pos in [8, 12]:
            return self.items[pos - 4], self.items[pos + 4],\
                   self.items[pos - 1]
        elif pos in [2, 3]:
            return self.items[pos - 1], self.items[pos + 1], self.items[pos + 4]
        elif pos in [14, 15]:
            return self.items[pos - 1], self.items[pos + 1],\
                  self.items[pos - 4]
        elif pos == 1:
            return self.items[pos + 1], self.items[pos + 4]
        elif pos == 4:
            return self.items[pos - 1], self.items[pos + 4]
        elif pos == 13:
            return self.items[pos + 1], self.items[pos - 4]
        elif pos == 16:
            return self.items[pos - 1], self.items[pos - 4]

    def game_over(self):
        flag = False
        for a, b in self.items.items():
            if b == '    ':
                pass
            else:
                if a == int(b.strip()):
                    flag = True
                else:
                    flag = False
        return flag


g = Puzzle()
g.build_board(int(input('Enter the difficulty : 0 1 2\n2 '
                        '=> highest 0=> lowest\n')))
g.main_frame()
print('Enter 0 to exit')
while True:
    print('Hello user:\nTo change the position just enter the no. near it')
    lst = g.valid_moves()
    lst1 = []
    for i in lst:
        lst1.append(int(i.strip()))
        print(i.strip(), '\t', end='')
    print()
    x = int(input())
    if x == 0:
        break
    elif x not in lst1:
        print('Wrong move')
    else:
        g.change(x)
    g.main_frame()
    if g.game_over():
        print('You WON')
        break
```

```
Enter the difficulty : 0 1 2
2 => highest 0=> lowest
1
+-----+-----+-----+-----+
|  1  |  7  |  2  |  4  |
+-----+-----+-----+-----+
|  5  |  6  |  3  |  12 |
+-----+-----+-----+-----+
|  9  |  8  |  11 |  15 |
+-----+-----+-----+-----+
|  13 |  10 |  14 |     |
+-----+-----+-----+-----+
```

```
Enter 0 to exit
Hello user:
To change the position just enter the no. near it
14      15
14
+-----+-----+-----+-----+
|  1  |  7  |  2  |  4  |
+-----+-----+-----+-----+
|  5  |  6  |  3  |  12 |
+-----+-----+-----+-----+
|  9  |  8  |  11 |  15 |
+-----+-----+-----+-----+
|  13 |  10 |     |  14 |
+-----+-----+-----+-----+
Hello user:
To change the position just enter the no. near it
10      14      11
```

Python: using tkinter

```python
''' Python 3.6.5 code using Tkinter graphical user interface.'''

from tkinter import *
from tkinter import messagebox
import random

# ********************************************

class Board:
    def __init__(self, playable=True):
        while True:
            # list of text for game squares:
            self.lot = [str(i) for i in range(1,16)] + ['']
            if not playable:
                break
            # list of text for game squares randomized:
            random.shuffle(self.lot)
            if self.is_solvable():
                break

        # game board is 2D array of game squares:
        self.bd = []
        i = 0
        for r in range(4):
            row = []
            for c in range(4):
                row.append(Square(r,c,self.lot[i]))
                i += 1
            self.bd.append(row)

    # How to check if an instance of a 15 puzzle
    # is solvable is explained here:
    # https://www.geeksforgeeks.org/check-instance-15-puzzle-solvable/
    # I only coded for the case where N is even.
    def is_solvable(self):
        inv = self.get_inversions()
        odd = self.is_odd_row()
        if inv % 2 == 0 and odd:
            return True
        if inv % 2 == 1 and not odd:
            return True
        return False

    def get_inversions(self):
        cnt = 0
        for i, x in enumerate(self.lot[:-1]):
            if x != '':
                for y in self.lot[i+1:]:
                    if y != '' and int(x) > int(y):
                        cnt += 1
        return cnt

    # returns True if open square is in odd row from bottom:
    def is_odd_row(self):
```

```
        idx = self.lot.index('')
        return idx in [4,5,6,7,12,13,14,15]

    # returns name, text, and button object at row & col:
    def get_item(self, r, c):
        return self.bd[r][c].get()

    def get_square(self, r, c):
        return self.bd[r][c]

    def game_won(self):
        goal = [str(i) for i in range(1,16)] + ['']
        i = 0
        for r in range(4):
            for c in range(4):
                nm, txt, btn = self.get_item(r,c)
                if txt != goal[i]:
                    return False
                i += 1
        return True

# ********************************************

class Square:       # ['btn00', '0', None]
    def __init__(self, row, col, txt):
        self.row = row
        self.col = col
        self.name = 'btn' + str(row) + str(col)
        self.txt = txt
        self.btn = None

    def get(self):
            return [self.name, self.txt, self.btn]

    def set_btn(self, btn):
        self.btn = btn

    def set_txt(self, txt):
        self.txt = txt

# ********************************************

class Game:
    def __init__(self, gw):
        self.window = gw

        # game data:
        self.bd = None
        self.playable = False

        # top frame:
        self.top_fr = Frame(gw,
                            width=600,
                            height=100,
                            bg='light green')
        self.top_fr.pack(fill=X)

        self.hdg = Label(self.top_fr,
                         text='  15 PUZZLE GAME  ',
                         font='arial 22 bold',
                         fg='Navy Blue',
                         bg='white')
        self.hdg.place(relx=0.5, rely=0.4,
                       anchor=CENTER)

        self.dir = Label(self.top_fr,
                 text="(Click 'New Game' to begin)",
                 font='arial 12 ',
                 fg='Navy Blue',
                 bg='light green')
        self.dir.place(relx=0.5, rely=0.8,
                       anchor=CENTER)

        self.play_btn = Button(self.top_fr,
                               text='New \nGame',
                               bd=5,
                               bg='PaleGreen4',
                               fg='White',
```

```
                                font='times 12 bold',
                                command=self.new_game)
        self.play_btn.place(relx=0.92, rely=0.5,
                            anchor=E)

        # bottom frame:
        self.btm_fr = Frame(gw,
                            width=600,
                            height=500,
                            bg='light steel blue')
        self.btm_fr.pack(fill=X)

        # board frame:
        self.bd_fr = Frame(self.btm_fr,
                           width=400+2,
                           height=400+2,
                           relief='solid',
                           bd=1,
                           bg='lemon chiffon')
        self.bd_fr.place(relx=0.5, rely=0.5,
                         anchor=CENTER)

        self.play_game()

# ********************************************

    def new_game(self):
        self.playable = True
        self.dir.config(text='(Click on a square to move it)')
        self.play_game()

    def play_game(self):
        # place squares on board:
        if self.playable:
            btn_state = 'normal'
        else:
            btn_state = 'disable'
        self.bd = Board(self.playable)
        objh = 100  # widget height
        objw = 100  # widget width
        objx = 0    # x-position of widget in frame
        objy = 0    # y-position of widget in frame

        for r in range(4):
            for c in range(4):
                nm, txt, btn = self.bd.get_item(r,c)
                bg_color = 'RosyBrown1'
                if txt == '':
                    bg_color = 'White'
                game_btn = Button(self.bd_fr,
                                  text=txt,
                                  relief='solid',
                                  bd=1,
                                  bg=bg_color,
                                  font='times 12 bold',
                                  state=btn_state,
                                  command=lambda x=nm: self.clicked(x))
                game_btn.place(x=objx, y=objy,
                               height=objh, width=objw)

                sq = self.bd.get_square(r,c)
                sq.set_btn(game_btn)

                objx = objx + objw
            objx = 0
            objy = objy + objh

    # processing when a square is clicked:
    def clicked(self, nm):
        r, c = int(nm[3]), int(nm[4])
        nm_fr, txt_fr, btn_fr = self.bd.get_item(r,c)

        # cannot 'move' open square to itself:
        if not txt_fr:
            messagebox.showerror(
                'Error Message',
                'Please select "square" to be moved')
            return
```

```
        # 'move' square to open square if 'adjacent' to it:
        adjs = [(r-1,c), (r, c-1), (r, c+1), (r+1, c)]
        for x, y in adjs:
            if 0 <= x <= 3 and 0 <= y <= 3:
                nm_to, txt_to, btn_to = self.bd.get_item(x,y)
                if not txt_to:
                    sq = self.bd.get_square(x,y)
                    sq.set_txt(txt_fr)
                    sq = self.bd.get_square(r,c)
                    sq.set_txt(txt_to)
                    btn_to.config(text=txt_fr,
                                  bg='RosyBrown1')
                    btn_fr.config(text=txt_to,
                                  bg='White')
                    # check if game is won:
                    if self.bd.game_won():
                        ans = messagebox.askquestion(
                            'You won!!!   Play again?')
                        if ans == 'no':
                            self.window.destroy()
                        else:
                            self.new_game()
                    return

        # cannot move 'non-adjacent' square to open square:
        messagebox.showerror(
            'Error Message',
            'Illigal move, Try again')
        return

# ********************************************

root = Tk()
root.title('15 Puzzle Game')
root.geometry('600x600+100+50')
root.resizable(False, False)
g = Game(root)
root.mainloop()
```

QB64

```
_TITLE "GUI Sliding Blocks Game "
RANDOMIZE TIMER

' get from user the desired board size = s
DO
    LOCATE CSRLIN, 3: INPUT "(0 quits) Enter your number of blocks per side 3 - 9 you want > ", s
    IF s = 0 THEN END
LOOP UNTIL s > 2 AND s < 10

' screen setup: based on the square blocks q pixels a sides
q = 540 / s 'square size, shoot for 540 x 540 pixel board display
SCREEN _NEWIMAGE(q * s + 1, q * s + 1, 32): _SCREENMOVE 360, 60

'initialize board = solution
DIM board(s, s)
FOR r = 1 TO s
    FOR c = 1 TO s
        board(c, r) = c + (r - 1) * s
    NEXT
NEXT
board(s, s) = 0: c0 = s: r0 = s

'scramble board for puzzle
FOR i = 0 TO s ^ 5 ' mix blocks
    SELECT CASE INT(RND * 4) + 1
        CASE 1: IF c0 < s THEN board(c0, r0) = board(c0 + 1, r0): board(c0 + 1, r0) = 0: c0 = c0 + 1
        CASE 2: IF c0 > 1 THEN board(c0, r0) = board(c0 - 1, r0): board(c0 - 1, r0) = 0: c0 = c0 - 1
        CASE 3: IF r0 < s THEN board(c0, r0) = board(c0, r0 + 1): board(c0, r0 + 1) = 0: r0 = r0 + 1
        CASE 4: IF r0 > 1 THEN board(c0, r0) = board(c0, r0 - 1): board(c0, r0 - 1) = 0: r0 = r0 - 1
    END SELECT
NEXT
```

```
t = TIMER: update = -1 'OK user here you go!
DO
    IF update THEN 'display status and determine if solved
        solved = -1: update = 0
        FOR r = 1 TO s
            FOR c = 1 TO s
                IF board(c, r) THEN
                    IF board(c, r) <> (r - 1) * s + c THEN solved = 0
                    COLOR _RGB32(255, 255, 255), _RGB32(0, 0, 255)
                    LINE ((c - 1) * q + 1, (r - 1) * q + 2)-(c * q - 2, r * q - 2), _RGB32(0, 0, 255), BF
                    _PRINTSTRING ((c - 1) * q + .4 * q, (r - 1) * q + .4 * q), RIGHT$(" " + STR$(board(c, r)), 2)
                ELSE
                    IF board(s, s) <> 0 THEN solved = 0
                    COLOR _RGB32(0, 0, 0), _RGB32(0, 0, 0)
                    LINE ((c - 1) * q, (r - 1) * q)-(c * q, r * q), , BF
                END IF
            NEXT
        NEXT
        IF solved THEN 'flash the Solved Report until user closes window else report status
            _DISPLAY
            flash$ = "Solved!" + STR$(mc) + " Moves in " + STR$(INT(TIMER - t)) + " secs."
            WHILE 1: _TITLE flash$: _DELAY .2: _TITLE "  ": _DELAY .2: WEND
        ELSE
            _TITLE STR$(mc) + " Moves in " + STR$(INT(TIMER - t)) + " secs." + STR$(test)
        END IF
        _DISPLAY
    END IF

    'get next mouse click, check if on block next to empty space make move or beep
    m = _MOUSEINPUT: mb = _MOUSEBUTTON(1): mx = _MOUSEX: my = _MOUSEY
    IF mb AND solved = 0 THEN 'get last place mouse button was down
        mb = _MOUSEBUTTON(1): mx = _MOUSEX: my = _MOUSEY
        WHILE mb 'left button down, wait for mouse button release
            m = _MOUSEINPUT: mb = _MOUSEBUTTON(1): mx = _MOUSEX: my = _MOUSEY
        WEND

        'convert mouse position to board array (x, y) are we near empty space?
        bx = INT(mx / q) + 1: by = INT(my / q) + 1: update = -1
        IF bx = c0 + 1 AND by = r0 THEN
            board(c0, r0) = board(c0 + 1, r0): board(c0 + 1, r0) = 0: c0 = c0 + 1: mc = mc + 1
        ELSEIF bx = c0 - 1 AND by = r0 THEN
            board(c0, r0) = board(c0 - 1, r0): board(c0 - 1, r0) = 0: c0 = c0 - 1: mc = mc + 1
        ELSEIF bx = c0 AND by = r0 + 1 THEN
            board(c0, r0) = board(c0, r0 + 1): board(c0, r0 + 1) = 0: r0 = r0 + 1: mc = mc + 1
        ELSEIF bx = c0 AND by = r0 - 1 THEN
            board(c0, r0) = board(c0, r0 - 1): board(c0, r0 - 1) = 0: r0 = r0 - 1: mc = mc + 1
        ELSE
            BEEP
        END IF
    END IF
LOOP
```

Quackery

The inputs are w,s,a,d and they refer to the direction in which the empty place "moves". After every prompt you can input a string of characters and submit them by pressing enter. Any characters other than w,s,a,d will be ignored, and moves will be performed according to the correct commands. Commands that would "move" the empty space off the edge of the puzzle are also ignored. For example, on a solved puzzle "asgw" would move 15 to the right, ignore 's' and 'g' and move 11 down.

```
( moves: 0 - up, 1 - down, 2 - left, 3 - right )

[ stack ]                           is 15.solver ( --> [ )

[ 0 swap find ]                     is find-empty ( board --> n )

[ over + dup -1 >
  over 4 < and iff
  [ nip true ] else
  [ drop false ] ] is try-nudge ( coord delta --> coord ? )
```

```
[ dip [ dup find-empty 4 /mod ]
  2 /mod 2 * 1 -
  swap iff try-nudge else
  [ dip swap
    try-nudge
    dip swap ]
  dip [ swap 4 * + ] ]              is try-move-target ( board move --> board target ? )

[ 2dup peek dip
  [ -1 unrot poke
    0 over find ]
  unrot poke
  0 swap -1 over find poke ]        is move-to ( board target --> board )

[ try-move-target
  iff [ move-to true ]
  else [ drop false ] ]             is try-move ( board move --> board ? )

[ [] 15 times
  [ i^ 1+ join ]
  0 join ]                  constant is <board> ( --> board )

[ <board>
  410 times ( 2 * the upper bound on the maximum moves required to solve )
  [ 4 random
    try-move drop ] ]               is <random-board> ( --> board )

[ peek dup iff
  [ number$
    dup size 1 =
    if [ space swap join ] ]
  else [ drop $ '  ' ]
  echo$ ]                           is echo-cell ( board n --> )

[ 4 * 4 times
  [ say '| '
    2dup i^ +
    echo-cell
    say ' ' ]
  say '|'
  2drop ]                           is echo-row ( board n --> )

[ 4 times
  [ say '+----+----+----+----+' cr
    dup i^ echo-row cr ]
  say '+----+----+----+----+' cr
  drop ]                            is echo-board ( board --> )

[ <board> = ]                       is solved? ( board --> ? )

[ $ 'Moves: ' input ]               is input-moves ( --> $ )

[ dup char w = iff
  [ drop 0 true ] done
  dup char s = iff
  [ drop 1 true ] done
  dup char a = iff
  [ drop 2 true ] done
  dup char d = iff
  [ drop 3 true ] done
  false ]                           is char-to-move ( c --> move true | c false )

[ input-moves witheach
  [ char-to-move
    if try-move
    drop ] ]                        is player-move ( board --> board )

[ ]'[ 15.solver put
  [ cr dup echo-board
    dup solved? not while
    15.solver share do
    again ]
  15.solver release drop ]          is 15-solve-with ( board 'solver --> board )

[ <random-board>
  15-solve-with player-move ]       is 15-puzzle ( --> )
```

```
15-puzzle
```

R

The inputs are w,a,s,d and they refer to the direction in which the adjacent piece moves into the empty space. For example, on a solved puzzle, "d" would move the 15 to the right.

```
puz15<-function(scramble.length=100){
  m=matrix(c(1:15,0),byrow=T,ncol=4)
  scramble=sample(c("w","a","s","d"),scramble.length,replace=T)
  for(i in 1:scramble.length){
    m=move(m,scramble[i])
  }
  win=F
  turn=0
  while(!win){
    print.puz(m)
    newmove=getmove()
    if(newmove=="w"|newmove=="a"|newmove=="s"|newmove=="d"){
      m=move(m,newmove)
      turn=turn+1
    }
    else{
      cat("Input not recognized","\n")
    }
    if(!(F %in% m==matrix(c(1:15,0),byrow=T,ncol=4))){
      win=T
    }
  }
  print.puz(m)
  cat("\n")
  print("You win!")
  cat("\n","It took you",turn,"moves.","\n")
}

getmove<-function(){
  direction<-readline(prompt="Move:")
  return(direction)
}
move<-function(m,direction){
  if(direction=="w"){
    m=move.u(m)
  }
  else if(direction=="s"){
    m=move.d(m)
  }
  else if(direction=="a"){
    m=move.l(m)
  }
  else if(direction=="d"){
    m=move.r(m)
  }
  return(m)
}
move.u<-function(m){
  if(0 %in% m[4,]){}
  else{
    pos=which(m==0)
    m[pos]=m[pos+1]
    m[pos+1]=0
  }
  return(m)
}
move.d<-function(m){
  if(0 %in% m[1,]){}
  else{
    pos=which(m==0)
    m[pos]=m[pos-1]
    m[pos-1]=0
  }
  return(m)
}
```

```
move.l<-function(m){
  if(0 %in% m[,4]){return(m)}
  else{return(t(move.u(t(m))))}
}
move.r<-function(m){
  if(0 %in% m[,1]){return(m)}
  else{return(t(move.d(t(m))))}
}
print.puz<-function(m){
  cat("+----+----+----+----+","\n")
  for(r in 1:4){
    string="|"
    for(c in 1:4){
      if(m[r,c]==0)
        string=paste(string,"    |",sep="")
      else if(m[r,c]<10)
        string=paste(string,"  ",m[r,c]," |",sep="")
      else
        string=paste(string," ",m[r,c]," |",sep="")
    }
    cat(string,"\n","+----+----+----+----+","\n",sep="")
  }
}
```

Sample output:

```
> puz15(scramble.length=4)
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 |    | 12 |
+----+----+----+----+
| 13 | 14 | 11 | 15 |
+----+----+----+----+
Move:w
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
| 13 | 14 |    | 15 |
+----+----+----+----+
Move:a
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
| 13 | 14 | 15 |    |
+----+----+----+----+

[1] "You win!"

 It took you 2 moves.
```

Racket

This is a GUI game; and there are difficulties getting screen shots onto RC. Use the arrow keys to slide the *blank* square.

It uses the `2htdp/universe` package.

```
#lang racket/base
(require 2htdp/universe 2htdp/image racket/list racket/match)
```

```
(define ((fifteen->pict (finished? #f)) fifteen)
  (for/fold ((i (empty-scene 0 0))) ((r 4))
    (define row
      (for/fold ((i (empty-scene 0 0))) ((c 4))
        (define v (list-ref fifteen (+ (* r 4) c)))
        (define cell
          (if v
              (overlay/align
               "center" "center"
               (rectangle 50 50 'outline (if finished? "white" "blue"))
               (text (number->string v) 30 "black"))
              (rectangle 50 50 'solid (if finished? "white" "powderblue"))))
        (beside i cell)))
    (above i row)))

(define (move-space fifteen direction)
  (define idx (for/first ((i (in-naturals)) (x fifteen) #:unless x) i))
  (define-values (row col) (quotient/remainder idx 4))
  (define dest (+ idx (match direction
                        ['l #:when (> col 0) -1]
                        ['r #:when (< col 3)  1]
                        ['u #:when (> row 0) -4]
                        ['d #:when (< row 3)  4]
                        [else 0])))
    (list-set (list-set fifteen idx (list-ref fifteen dest)) dest #f))

(define (key-move-space fifteen a-key)
  (cond [(key=? a-key "left") (move-space fifteen 'l)]
        [(key=? a-key "right") (move-space fifteen 'r)]
        [(key=? a-key "up") (move-space fifteen 'u)]
        [(key=? a-key "down") (move-space fifteen 'd)]
        [else fifteen]))

(define (shuffle-15 fifteen shuffles)
  (for/fold ((rv fifteen)) ((_ shuffles))
    (move-space rv (list-ref '(u d l r) (random 4)))))

(define fifteen0 '(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 #f))

(define (solved-world? w) (equal? w fifteen0))

(big-bang (shuffle-15 fifteen0 200)
          (name "Fifteen")
          (to-draw (fifteen->pict))
          (stop-when solved-world? (fifteen->pict #t))
          (on-key key-move-space))
```

Raku

(formerly Perl 6)

Works with: Rakudo version 2018.06

Most of this is interface code. Reused substantial portions from the 2048 task. Use the arrow keys to slide tiles, press 'q' to quit or 'n' for a new puzzle. Requires a POSIX termios aware terminal. Ensures that the puzzle is solvable by shuffling the board with an even number of swaps, then checking for even taxicab parity for the empty space.

```
use Term::termios;

constant $saved   = Term::termios.new(fd => 1).getattr;
constant $termios = Term::termios.new(fd => 1).getattr;
# raw mode interferes with carriage returns, so
# set flags needed to emulate it manually
$termios.unset_iflags(<BRKINT ICRNL ISTRIP IXON>);
$termios.unset_lflags(< ECHO ICANON IEXTEN ISIG>);
$termios.setattr(:DRAIN);

# reset terminal to original setting on exit
END { $saved.setattr(:NOW) }
```

```
13
14  constant n    = 4; # board size
15  constant cell = 6; # cell width
16
17  constant $top = join '─' x cell, '┌', '┬' xx n-1, '┐';
18  constant $mid = join '─' x cell, '├', '┼' xx n-1, '┤';
19  constant $bot = join '─' x cell, '└', '┴' xx n-1, '┘';
20
21  my %dir = (
22     "\e[A" => 'up',
23     "\e[B" => 'down',
24     "\e[C" => 'right',
25     "\e[D" => 'left',
26  );
27
28  my @solved = [1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,' '];
29  my @board;
30  new();
31
32  sub new () {
33      loop {
34         @board = shuffle();
35         last if parity-ok(@board);
36     }
37  }
38
39  sub parity-ok (@b) {
40      my $row = @b.first(/' '/,:k);
41      my $col = @b[$row].first(/' '/,:k);
42      so ([3,3] <<->> [$row,$col]).sum %% 2;
43  }
44
45  sub shuffle () {
46      my @c = [1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,' '];
47      for (^16).pick(*) -> $y, $x {
48          my ($yd, $ym, $xd, $xm) = ($y div n, $y mod n, $x div n, $x mod n);
49          @(@c[$ym;$yd],@c[$xm;$xd]) = @(@c[$xm;$xd],@c[$ym;$yd]);
50      }
51      @c;
52  }
53
54  sub row (@row) { '│' ~ (join '│', @row».&center) ~ '│' }
55
56  sub center ($s){
57      my $c   = cell - $s.chars;
58      my $pad = ' ' x ceiling($c/2);
59      sprintf "%{cell}s", "$s$pad";
60  }
61
62  sub draw-board {
63      run('clear');
64      print qq:to/END/;
65
66
67      Press direction arrows to move.
68
69      Press q to quit. Press n for a new puzzle.
70
71      $top
72      { join "\n\t$mid\n\t", map { .&row }, @board }
73      $bot
74
75      { (so @board ~~ @solved) ?? 'Solved!!' !! '' }
76  END
77  }
78
79  sub slide (@c is copy) {
80      my $t = (grep { /' '/ }, :k, @c)[0];
81      return @c unless $t and $t > 0;
82      @c[$t,$t-1] = @c[$t-1,$t];
83      @c;
84  }
85
86  proto sub move (|) {*};
87
88  multi move('up') {
89      map { @board[*;$_] = reverse slide reverse @board[*;$_] }, ^n;
90  }
```

```
91
92  multi move('down') {
93      map { @board[*;$_] = slide @board[*;$_] }, ^n;
94  }
95
96  multi move('left') {
97      map { @board[$_] = reverse slide reverse @board[$_] }, ^n;
98  }
99
100 multi move('right') {
101     map { @board[$_] = slide @board[$_] }, ^n;
102 }
103
104 loop {
105     draw-board;
106
107     # Read up to 4 bytes from keyboard buffer.
108     # Page navigation keys are 3-4 bytes each.
109     # Specifically, arrow keys are 3.
110     my $key = $*IN.read(4).decode;
111
112     move %dir{$key} if so %dir{$key};
113     last if $key eq 'q'; # (q)uit
114     new() if $key eq 'n';
115 }
```

Sample screen shot:

```
        Press direction arrows to move.

        Press q to quit. Press n for a new puzzle.

        ┌────┬────┬────┬────┐
        │ 2  │ 1  │ 10 │ 14 │
        ├────┼────┼────┼────┤
        │ 15 │ 11 │ 12 │    │
        ├────┼────┼────┼────┤
        │ 13 │ 3  │ 6  │ 7  │
        ├────┼────┼────┼────┤
        │ 9  │ 4  │ 5  │ 8  │
        └────┴────┴────┴────┘
```

Rebol

```
rebol []  random/seed now  g: [style t box red [
  if not find [0x108 108x0 0x-108 -108x0] face/offset - e/offset [exit]
  x: face/offset face/offset: e/offset e/offset: x] across
] x: random repeat i 15 [append x:[] i]  repeat i 15 [
  repend g ['t mold x/:i random white] if find [4 8 12] i [append g 'return]
] append g [e: box]  view layout g
```

Red

```
Red [Needs: 'view]
system/view/screens/1/pane/-1/visible?: false ;; suppress console window
;; because this code is heavily influenced / (copied from ... )
;; by the rebol version ( which actually works with red as well with
;; some minimal changes ) i added some comments to make the
;; code more readable...

;;----------------------------------------
;; check, if puzzle is solved, OK button closes program
;;----------------------------------------
check: does [
;; the text on each tile must be equal to its position:
  repeat pos 15 [ unless pos = to-integer lay/pane/:pos/text  [exit] ]
  view/flags [  title "SuCCeSS"
                text "You solved the puzzle ! "  button "OK"  [quit]
```

```
    ] [modal popup]
]
;;--------------------------------------
;; actually only changes text / Number on tile / button
;;--------------------------------------
changeTiles: func [ f ][
  ;; this is tricky, check if e(mpty) button is one in of the
  ;; four valid positions , by simply subtracting their offsets...else exit ( no change)
  unless find  [ 0x52 0x-52 52x0 -52x0 ] f/offset - e/offset [return false]
  e/text: f/text  ;; empty button gets Number from clicked button
  e/visible?: true  ;; empty gets visible
  f/visible?: false ;; clicked button gets invisible
  e: f    ;; e(mpty) refers now to clicked button
  return true ;; true - change has happened
]
;;--------------------------------------
;; function which is executed, when button is clicked
;; argument is face - button
;;--------------------------------------
bClicked: func [f][   if changeTiles f [check] ]

;; define the window:
;; style is a prototype definition for a button element
win:   [ title "15 Puzzle"
        backdrop silver
        style btn: button 40x40  [bClicked face] font-size 15 bold
]
;; generate 1..15 buttons with number
repeat nr 15 [
     repend win  [ 'btn form nr ] ;; repend reduces and then appends..
     if  0 = mod nr 4  [  append win  'return ] ;; "carriage return" in window after 4 buttons
    ]
  ;; append empty / hidden button no 16 and return/quit button:
    append win [ e: btn "16" hidden return  button "Quit" [quit]]

lay: layout win
;; define win as layout, so  each button can now be addressed as
;; layout/pane/index ( 1..15 - redbol is 1 based )

flip: 0
;; start random generator, otherwise we always get the same puzzle
random/seed now/time/precise
;; lets flip random tiles a few times to initialize the puzzle
;; ( actually numbers < 100 are quite easy to solve )

while [ flip < 40 ] [ if changeTiles lay/pane/(random 16)  [ flip:  flip  + 1 ]  ]

;; show the window...
view  lay
```

REXX

This REXX version allows the user to specify the size of the puzzle (**N**, where **NxN** is the size of the puzzle).

With some more complexity, the REXX computer program could be changed to allow multiple─tile moves (so that,
for instance, three tiles could be slid to the right).

Over half of the REXX program has to do with input validation and presentation of the puzzle (grid).

```
/*REXX pgm implements the 15─puzzle (AKA: Gem Puzzle, Boss Puzzle, Mystic Square, 14─15)*/
parse arg N seed .                              /*obtain optional arguments from the CL*/
if N=='' | N=="," then N=4                      /*Not specified?  Then use the default.*/
if datatype(seed, 'W') then call random ,,seed  /*use repeatability seed for RANDOM BIF*/
nh= N**2;   @.=;    nn= nh - 1;    w= length(nn) /*define/initialize some handy values. */
$=                                              /*$: will hold the solution for testing*/
       do i=1  for nn;  $= $  i                 /* [◄]  build a solution for testing.  */
       end   /*i*/
```

```
done= $                                          /* [↓]  scramble the tiles in puzzle.  */
      do j=1  for nn;  a= random(1, words($) );    @.j= word($, a);   $= delword($, a, 1)
      end   /*j*/
                          /*═══════════════════ play the 15─puzzle 'til done or quit.*/
   do  until puzz==done & @.nh==''               /*perform moves until puzzle is solved.*/
   call getmv                                    /*get user's move(s)  and  validate it.*/
   if errMsg\==''  then do;  say sep errMsg;       iterate       /*possible error msg? */
                        end
   call showGrid 0                               /*don't display puzzle, just find hole.*/
   if wordpos(x, !)==0  then do;  say sep  'tile '     x     " can't be moved.";   iterate
                             end
   @.hole= x;    @.tile=
   call showGrid 0                               /*move specified tile ──► puzzle hole.*/
   end   /*until*/           /*═══════════════════════════════════════════════════════*/

call showGrid 1;    say;     say sep  'Congratulations!   The'      nn"-puzzle is solved."
exit 0                                           /*stick a fork in it,  we're all done. */
/*──────────────────────────────────────────────────────────────────────────────────────*/
getmv: x= 0;   sep= copies('─', 8);     pad= left('', 1 + length(sep) )   /*pad=9 blanks*/
       prompt= sep    'Please enter a tile number  or  numbers '     sep     " (or Quit)."
       if queued()==0  then do;    say;    call showGrid 1;     say;      say prompt
                            end
       parse pull x . 1 ox . 1 . zx;   upper x   /*obtain a number (or numbers) from CL.*/
       if abbrev('QUIT', x, 1)  then do;   say;   say;    say sep  "quitting.";      exit
                                     end
       if words(zx)>0  then do;  parse var  zx    xq;     queue xq
                            end                 /* [↑]  Extra moves?  Stack for later. */
             select                           /* [↓]  Check for possible errors/typos*/
             when x==''              then errMsg= "nothing entered."
             when \datatype(x, 'N')  then errMsg= "tile number isn't numeric: "        ox
             when \datatype(x, 'W')  then errMsg= "tile number isn't an integer: "     ox
             when x=0                then errMsg= "tile number can't be zero: "        ox
             when x<0                then errMsg= "tile number can't be negative: "    ox
             when x>nn               then errMsg= "tile number can't be greater than"  nn
             otherwise                    errMsg=
             end   /*select*/                  /* [↑]  verify the human entered data. */
       return
/*──────────────────────────────────────────────────────────────────────────────────────*/
showGrid: parse arg show;       !.=;                    x= x/1;       #= 0;        puzz=
          top= '╔'copies( copies("═", w)'╦', N);       top= left( top, length(top) -1)"╗"
          bar= '╠'copies( copies("═", w)'╬', N);       bar= left( bar, length(bar) -1)"╣"
          bot= '╚'copies( copies("═", w)'╩', N);       bot= left( bot, length(bot) -1)"╝"
          if show  then say pad top

               do   r=1  for N;     z= '║'
                 do c=1  for N;     #= #+1;     y= @.#;      puzz= puzz y;        !.r.c= y
                 _= right(@.#, w)"║";           z= z || _              /* [↓]  find hole*/
                 if @.# == ''  then do;    hole= #;     holeRow= r;     holeCol= c;    end
                 if @.# == x   then do;    tile= #;     tileRow= r;     tileCol= c;    end
                 end   /*c*/                                           /* [↑]  find  X. */
               if show  then do;    say pad z;     if r\==N  then say pad bar;         end
               end     /*r*/

          rm=holeRow-1;   rp=holeRow+1;  cm=holeCol-1;  cp=holeCol+1   /*possible moves.*/
          !=!.rm.holeCol  !.rp.holeCol   !.holeRow.cm   !.holeRow.cp   /* legal   moves.*/
          if show  then say pad bot;                  return
```

output when using the default input:

```
          ╔══╦══╦══╦══╗
          ╠10╬ 7╬ 8╬11╣
          ╠══╬══╬══╬══╣
          ╠ 4╬ 3╬15╬ 1╣
          ╠══╬══╬══╬══╣
          ╠ 9╬12╬ 2╬13╣
          ╠══╬══╬══╬══╣
          ╠14╬ 5╬ 6╬  ╣
          ╚══╩══╩══╩══╝


──────── Please enter a tile number  or  numbers  ────────  (or Quit).
13                          ◄■■■■■■■■■■ user input.
          ╔══╦══╦══╦══╗
          ╠10╬ 7╬ 8╬11╣
          ╠══╬══╬══╬══╣
          ╠ 4╬ 3╬15╬ 1╣
```

```
            ╔══╦══╦══╦══╗
            ╠══╬══╬══╬══╣
            ║ 9║12║ 2║  ║
            ╠══╬══╬══╬══╣
            ╠══╬══╬══╬══╣
            ║14║ 5║ 6║13║
            ╚══╩══╩══╩══╝

──────── Please enter a tile number  or  numbers  ────────  (or Quit).
1  15  3                        ◄■■■■■■■■■■ user input.

            ╔══╦══╦══╦══╗
            ╠══╬══╬══╬══╣
            ║10║ 7║ 8║11║
            ╠══╬══╬══╬══╣
            ║ 4║  ║ 3║15║
            ╠══╬══╬══╬══╣
            ║ 9║12║ 2║ 1║
            ╠══╬══╬══╬══╣
            ╠══╬══╬══╬══╣
            ║14║ 5║ 6║13║
            ╚══╩══╩══╩══╝

──────── Please enter a tile number  or  numbers  ────────  (or Quit).
quit                           ◄■■■■■■■■■■ user input.


──────── quitting.
```

Ring

```
# Project : CalmoSoft Fifteen Puzzle Game
# Date    : 2018/12/01
# Author  : Gal Zsolt (CalmoSoft), Bert Mariani
# Email   : calmosoft@gmail.com

load "guilib.ring"

app1 = new qapp {

        stylefusionblack()
        empty = 16
        nr_moves = 0
        nr_sleep = 1
        button_size = 4
        current_button = 4
        temp = 0
        flag_init = 0
        flag_save = 0
        flag_move = 0
        button = list(52)
        size_button = list(7)
        table1 = []
        table2 = []
        table3 = []
        n_degree = 0
        nr_degree = [0,90,180,270 ,-90,-180,-270]
        n_degreeRight = 0
        n_degreeLeft = 0
        btn_degree = newlist(52,2)
        counter_man = 0
        t1 = 0

        win1 = new qwidget() {
                   move(0,0)
                   resize(380,760)
                   setwindowtitle("CalmoSoft Fifteen Puzzle Game")

                  for n=1 to 52
                        for m=1 to 2
                             btn_degree[n][m] = 0
                        next
                  next

                   for n = 4 to 7
                               size_button[n] = new qpushbutton(win1)
                               {
                                                  col = n%4
                                                  setgeometry(100+col*40,60,40,40)
```

```
                                    settext(string(n) + "x" + string(n))
                                    setclickevent("newsize(" + string(n) + ")")
                }
     next

    btnMoves = new qpushbutton(win1)
    {
                  setgeometry(100,260,80,40)
                  settext("0")
                  show()
    }

     scramblebtn = new qpushbutton(win1)
     {
                          setgeometry(100,300,160,40)
                          settext("Scramble")
                          setclickevent("scramble()")
     }

     resetbtn = new qpushbutton(win1)
     {
                     setgeometry(100,340,160,40)
                     settext("Reset")
                     setclickevent("resettiles()")
     }

     savebtn = new qpushbutton(win1)
     {
                    setgeometry(100,380,160,40)
                    settext("Save Game")
                    setclickevent("pSave()")
     }

     playbtn = new qpushbutton(win1)
     {
                    setgeometry(100,420,160,40)
                    settext("Resume Game")
                    setclickevent("pPlay()")
     }

     sleepbtn = new qpushbutton(win1)
     {
                    setgeometry(100,460,160,40)
                    settext("Sleep Time: ")

     }

    decbtn = new qpushbutton(win1)
    {
                 setgeometry(220,460,40,40)
                 settext("<-")
                 setclickevent("pDecSleep()")
    }

    incbtn = new qpushbutton(win1)
    {
                 setgeometry(260,460,40,40)
                 settext("->")
                 setclickevent("pIncSleep()")
    }

   rightbtn = new qpushbutton(win1)
   {
                 setgeometry(100,500,160,40)
                 settext("In the Right Place : ")
   }

   timebtn = new qpushbutton(win1)
   {
                 setgeometry(100,540,160,40)
                 settext("Elapsed Time : ")
   }

  TimerMan = new qtimer(win1)
 {
                    setinterval(500)
                    settimeoutevent("pTime()")
                    stop()
```

```
                }
               newsize(4)
               show()
         }
         exec()
}

Func newlist x, y
     if isstring(x) x=0+x ok
     if isstring(y) y=0+y ok
     alist = list(x)
     for t in alist
         t = list(y)
     next
     return alist

func scramble
       for n= 1 to 1000
            current_button=random(button_size*button_size-1)+1
            up = (empty = (current_button - button_size))
            down = (empty = (current_button + button_size))
            left = ((empty = (current_button - 1)) and ((current_button % button_size) != 1))
            right = ((empty = (current_button + 1)) and ((current_button % button_size) != 0))
            move = up or down or left  or right
            if move = 1
               button[current_button] { temp2 = text() }
               col = empty%button_size
               if col = 0 col = button_size ok
               row = ceil(empty/button_size)
               button[empty] {
                       setgeometry(60+col*40,60+row*40,40,40)
                       rnd = random(6)+1
                       n_degree = nr_degree[rnd]
                       button[empty].setbuttoncolor("yellow")
                       button[empty].settext(temp2)
                       button[empty].setClickEvent("movetile(" + string(empty) +")")
                       btn_degree[empty] [1] = temp2
                       btn_degree[empty] [2] = n_degree
                       }
               button[current_button].setbuttoncolor("yellow")
               btn_degree[current_button][2] = 0
               button[current_button]{settext("")}
               empty = current_button
            ok
       next
       button[button_size*button_size+2]{settext("Here")}
       for n=1 to button_size*button_size
             button[n].setbuttoncolor("yellow")
       next
       table1 = []
       table2 = []
       table3 = []
       for n = 1 to button_size*button_size
             add(table1, button[n].text())
             add(table2, button[n].text())
             add(table3, string(btn_degree[n][2]))
       next
       add(table1, string(empty))
       add(table2, string(empty))
       add(table3, string(empty))
       add(table1, "OK")
       add(table2, "OK")
       add(table3, "OK")
       flag_save = 0
       flag_move = 0
       nr_moves = 0
       btnMoves.settext(string(nr_moves))
       timebtn.settext("Elapsed Time : ")
       t1 = clock()
       rightPlace()
       return

func movetile current_button2
       if (current_button2 = button_size*button_size-1 and button[current_button2].text() = "In")
           pBack()
       else
           see char(7)
           up = (empty = (current_button2 - button_size))
```

```
            down = (empty = (current_button2 + button_size))
            left = ((empty = (current_button2- 1)) and ((current_button2 % button_size) != 1))
            right = ((empty = (current_button2 + 1)) and ((current_button2 % button_size) != 0))
            move = up or down or left  or right
            if move = 1
               temp2 = button[current_button2].text()
               btn_degree[empty][1] = temp2
               add(table1, temp2)
               add(table2, string(current_button2))
               col = empty%button_size
               if col = 0 col = button_size ok
               row = ceil(empty/button_size)
               button[empty] {
                                setgeometry(60+col*40,60+row*40,40,40)
                                n_degree = btn_degree[current_button2][2]
                                btn_degree[empty][2] = n_degree
                                button[empty].setbuttoncolor("orange")
                                button[empty].settext(temp2)
               }
               add(table3, string(n_degree))
               button[current_button2].setbuttoncolor("cyan")
               button[current_button2]{settext("")}
               empty = current_button2
               nr_moves = nr_moves + 1
               btnMoves.settext(string(nr_moves))
               isGameOver()
            ok
         ok
         flag_move = 1
         pElapsedTime()
         rightPlace()
         return

func resettiles
           n_degree = 0
           empty = button_size*button_size
           for empty = 1 to button_size*button_size-1
                btn_degree[empty][2] = 0
                n_degree = 0
                btn_degree[empty][1] = string(empty)
                button[empty].setstylesheet("background-color:yellow")
                button[empty] {settext(string(empty))}
           next
           button[button_size*button_size].setstylesheet("background-color:yellow")
           button[button_size*button_size] {settext("")}
           table1 = []
           table2 = []
           table3 = []
           for n = 1 to button_size*button_size
                add(table1, button[n].text())
                add(table2, button[n].text())
                add(table3, string(btn_degree[n][2]))
           next
           add(table1, string(empty))
           add(table2, string(empty))
           add(table3, string(empty))
           add(table1, "OK")
           add(table2, "OK")
           add(table3, "OK")
           flag_save = 0
           flag_move = 0
           nr_moves = 0
           btnMoves.settext(string(nr_moves))
           timebtn.settext("Elapsed Time : ")
           t1 = clock()
           rightPlace()
           return

func pHere
           if button[button_size*button_size-1].text() != "" and button[button_size*button_size+2].text() = "Here"
              button[button_size*button_size-1] { temp = text() }
              button[button_size*button_size+2].close()
              button[button_size*button_size+2] = new ButtonWithRotatedText(win1)
              button[button_size*button_size+2] {
              setgeometry(60+(button_size-1)*40,60+(button_size+1)*40,40,40)
              setstylesheet("background-color:yellow")
              btn_degree[button_size*button_size+2][2] = btn_degree[button_size*button_size-1][2]
              n_degree = btn_degree[button_size*button_size+2][2]
```

```
            emptysave = empty
            empty = button_size*button_size+2
            btn_degree[empty][1] = temp
            settext(temp)
            }
            n_degree = 0
            empty = button_size*button_size-1
            btn_degree[empty][1] = "In"
            button[button_size*button_size-1]{settext("In")}
            for n = 1 to button_size*button_size
        button[n].setenabled(false)
            next
            button[button_size*button_size-1].setenabled(true)
            scramblebtn.setenabled(false)
            resetbtn.setenabled(false)
            savebtn.setenabled(false)
            playbtn.setenabled(false)
            empty = emptysave
         ok

func pBack
         button[button_size*button_size+2] { temp = text() }
         n_degree = btn_degree[button_size*button_size+2][2]
         btn_degree[button_size*button_size-1][2] = btn_degree[button_size*button_size+2][2]
         emptysave = empty
         empty = button_size*button_size-1
         btn_degree[empty][1] = temp
         button[button_size*button_size-1] {settext(temp)}
         button[button_size*button_size+2].close()
         button[button_size*button_size+2] = new qpushbutton(win1)
                 {
                 setgeometry(60+(button_size-1)*40,60+(button_size+1)*40,40,40)
                 settext("Here")
                 setclickevent("pHere()")
                 show()
                 }
         for n = 1 to button_size*button_size
               button[n].setenabled(true)
         next
         scramblebtn.setenabled(true)
         resetbtn.setenabled(true)
         savebtn.setenabled(true)
         playbtn.setenabled(true)
         empty = emptysave
         isGameOver()

func rotateleft
         if button[button_size*button_size+2].text() != "Here"
            button[button_size*button_size+2].close()
            button[button_size*button_size+2] = new ButtonWithRotatedText(win1)
                        button[button_size*button_size+2] {
                        setgeometry(60+(button_size-1)*40,60+(button_size+1)*40,40,40)
                        setstylesheet("background-color:yellow")
                        n_degreeLeft = (n_degreeLeft-90)%360
                        n_degree = n_degreeLeft
                        btn_degree[button_size*button_size+2][2] = n_degree
                        emptysave = empty
                        empty = button_size*button_size+2
                        btn_degree[empty][1] = temp
                        button[button_size*button_size+2]{settext(temp)}
                        }
                        empty = emptysave
         ok

func rotateright
         if button[button_size*button_size+2].text() != "Here"
            button[button_size*button_size+2].close()
            button[button_size*button_size+2] = new ButtonWithRotatedText(win1)
                        button[button_size*button_size+2] {
                        setgeometry(60+(button_size-1)*40,60+(button_size+1)*40,40,40)
                        setstylesheet("background-color:yellow")
                        n_degreeRight = (n_degreeRight+90)%360
                        n_degree = n_degreeRight
                        btn_degree[button_size*button_size+2][2] = n_degree
                        emptysave = empty
                        empty = button_size*button_size+2
                        btn_degree[empty][1] = temp
                        button[button_size*button_size+2]{settext(temp)}
```

```
                    }
                    empty = emptysave
        ok

func newsize current_button
        win1{
                sizenew = current_button%4
                win1.resize(360+sizenew*40,640+sizenew*40)
                if flag_init != 0
                   for nb = 1 to button_size*button_size+3
                         button[nb] {close()}
                   next
                   btnMoves.close()
                ok
                scramblebtn.close()
                resetbtn.close()
                savebtn.close()
                playbtn.close()
                btnMoves.close()
                sleepbtn.close()
                decbtn.close()
                incbtn.close()
                rightbtn.close()
                timebtn.close()

                for n = 1 to current_button*current_button
                     col = n%current_button
                     if col = 0 col = current_button ok
                     row = ceil(n/current_button)
                     button[n] = new ButtonWithRotatedText(win1)
                                 button[n] {
                                 setgeometry(60+col*40,60+row*40,40,40)
                                 button[n].setbuttoncolor("yellow")
                                 n_degree = 0
                                 if n < current_button*current_button
                                    button[n].settext(string(n))
                                 but n = current_button*current_button
                                    button[n].settext("")
                                 ok
                                 setClickEvent("movetile(" + string(n) +")")
                                 }
                next

                btnMoves = new qpushbutton(win1)
                {
                                  setgeometry(100,60+(current_button+1)*40,(current_button-3)*40,40)
                                  setStyleSheet("text-align:center")
                                  settext("0")
                                  show()
                }

                button[current_button*current_button+1] = new qpushbutton(win1)
                {
                                       setgeometry(60+(current_button-2)*40,60+(current_button+1)*40,40,40)
                                       settext("<-")
                                       setclickevent("rotateLeft()")
                                       show()
                }

                button[current_button*current_button+2] = new qpushbutton(win1)
                {
                                      setgeometry(60+(current_button-1)*40,60+(current_button+1)*40,40,40)
                                      settext("Here")
                                      setclickevent("pHere()")
                                      show()
                }

                button[current_button*current_button+3] = new qpushbutton(win1)
                {
                                      setgeometry(60+current_button*40,60+(current_button+1)*40,40,40)
                                      settext("->")
                                      setclickevent("rotateRight()")
                                      show()
                 }

                scramblebtn = new qpushbutton(win1)
                {
                                  setgeometry(100,100+(current_button+1)*40,current_button*40,40)
```

```
                        settext("Scramble")
                        setclickevent("scramble()")
                        show()
    }

    resetbtn = new qpushbutton(win1)
    {
                    setgeometry(100,100+(current_button+2)*40,current_button*40,40)
                    settext("Reset")
                    setclickevent("resettiles()")
                    show()
    }

    savebtn = new qpushbutton(win1)
    {
                    setgeometry(100,100+(current_button+3)*40,current_button*40,40)
                    settext("Save Game")
                    setclickevent("pSave()")
                    show()
    }

    playbtn = new qpushbutton(win1)
    {
                   setgeometry(100,100+(current_button+4)*40,current_button*40,40)
                   settext("Resume Game")
                   setclickevent("pPlay()")
                   show()
    }

    sleepbtn = new qpushbutton(win1)
    {
                   setgeometry(100,100+(current_button+5)*40,(current_button-2)*40,40)
                   settext("Sleep Time: " + string(nr_sleep) + " s")
                   show()
    }

    decbtn = new qpushbutton(win1)
    {
                   setgeometry(100+(current_button-2)*40,100+(current_button+5)*40,40,40)
                   settext("<-")
                   setclickevent("pDecSleep()")
                   show()
    }

    incbtn = new qpushbutton(win1)
    {
                   setgeometry(100+(current_button-1)*40,100+(current_button+5)*40,40,40)
                   settext("->")
                   setclickevent("pIncSleep()")
                   show()
    }

   rightbtn = new qpushbutton(win1)
    {
                   setgeometry(100,100+(current_button+6)*40,current_button*40,40)
                   settext("In the Right Place : ")
                   show()
    }

    timebtn = new qpushbutton(win1)
    {
                   setgeometry(100,100+(current_button+7)*40,current_button*40,40)
                   settext("Elapsed Time : ")
                   show()
    }

    table1 = []
    table2 = []
    table3 = []
    for n = 1 to button_size*button_size
          add(table1, button[n].text())
          add(table2, button[n].text())
          add(table3, string(0))
    next
    add(table1, string(empty))
    add(table2, string(empty))
    add(table3, string(empty))
    add(table1, "OK")
```

```
                add(table2, "OK")
                add(table3, "OK")
                empty = current_button*current_button
                button_size = current_button
                flag_init = 1
                flag_save = 0
                flag_move = 0
                nr_moves = 0
                timebtn.settext("Elapsed Time : ")
                t1 = clock()
                scramble()
                }

func pSave
        textedit1 = list2str(table1)
        textedit2 = list2str(table2)
        textedit3 = list2str(table3)
        chdir(currentdir())
        cName1 = "CalmoSoftPuzzle1.txt"
        cName2 = "CalmoSoftPuzzle2.txt"
        cName3 = "CalmoSoftPuzzle3.txt"
        write(cName1,textedit1)
        write(cName2,textedit2)
        write(cName3,textedit3)
         flag_save = 1
         timebtn.settext("Elapsed Time : ")
         t1 = clock()
         return

func pPlay
        if  flag_save = 0 or flag_move = 0
            warning()
        else
           chdir(currentdir())
           cName1 = "CalmoSoftPuzzle1.txt"
           textedit1 = read(cName1)
           table1 = str2list(textedit1)
           cName2 = "CalmoSoftPuzzle2.txt"
           textedit2 = read(cName2)
           table2 = str2list(textedit2)
           cName3 = "CalmoSoftPuzzle3.txt"
           textedit3 = read(cName3)
           table3 = str2list(textedit3)
           for empty = 1 to button_size*button_size
                button[empty].setbuttoncolor("yellow")
                n_degree = number(table3[empty])
                btn_degree[empty][1] = table1[empty]
                button[empty] {settext(table1[empty])}
           next
           empty = number(table1[button_size*button_size + 1])
           counter_man = button_size*button_size+2
           nr_moves = 0
           t1 = clock()
           TimerMan.start()
       ok

func pTime()
        if flag_save = 0 or flag_move = 0
           warning()
        else
           counter_man++
           pPlaySleep()
           sleep(nr_sleep*1000)
           pElapsedTime()
           if counter_man = len(table1)
              TimerMan.stop()
           ok
        ok

func pPlaySleep
        see char(7)
        value = table1[counter_man]
        place = table2[counter_man]
        n_degree = number(table3[counter_man])
        btn_degree[empty][1] = value
        button[empty].setbuttoncolor("orange")
        button[empty] {settext(value)}
        n_degree = 0
```

```
        button[number(place)].setbuttoncolor("cyan")
        button[number(place)] {settext("")}
        empty = number(place)
        nr_moves = nr_moves + 1
        btnMoves.settext(string(nr_moves))

func pIncSleep
        nr_sleep = nr_sleep + 1
        sleepbtn.settext("Sleep Time: " + string(nr_sleep) + " s")

func pDecSleep
        if nr_sleep > 1
           nr_sleep = nr_sleep - 1
           sleepbtn.settext("Sleep Time: " + string(nr_sleep) + " s")
        ok

func sleep(x)
        nTime = x
        oTest = new qTest
        oTest.qsleep(nTime)
        return

func isGameOver
        flagend = 1
        for n=1 to button_size*button_size-1
              if button[n].text() != n or btn_degree[n][2] != 0
                 flagend = 0
                 exit
              ok
        next
        if flagend = 1
           new qmessagebox(win1) {
                   setwindowtitle("Game Over")
                   settext("Congratulations!")
                   show()
                   }
        ok

func rightPlace
        count = 0
        for n=1 to button_size*button_size
             if button[n].text() = n and btn_degree[n][2] = 0
                count = count + 1
             ok
        next
        rightbtn.settext("In the Right Place : " + count)

func warning
        new qmessagebox(win1) {
                setwindowtitle("Warning!")
                settext("First you must play and save the game.")
                show()
                }

func pElapsedTime
        t2 = (clock() - t1)/1000
        timebtn.settext("Elapsed Time : " + t2 + " s")

Class ButtonWithRotatedText

         oButton oLabel  cText="We are here"  n_degree = 30  nTransX = 50   nTransY = 0

func init( oParent)
        oButton = new qPushButton(oParent)
        oLabel  = new qLabel(oParent)
        oLabel.setAttribute(Qt_WA_TransparentForMouseEvents,True)
        oLabel.setAttribute(Qt_WA_DeleteOnClose, True)
        oButton.setAttribute(Qt_WA_DeleteOnClose, True)
        oButton.Show()
        return

func close()
        oLabel.close()
        oButton.close()
        return

func setstylesheet(x)
        oButton.setstylesheet(x)
```

```
func setgeometry( x,y,width,height)
        oButton.setgeometry(x,y,width,height)
        oLabel.setgeometry( x,y,width,height)

func setText( cValue)
        cText = cValue
        return

func Text()
         return cText

func setTranslate( x,y )
         nTransX = x
         nTransY = y
         return

func TranslateOffsetX()
        return nTransX

func TranslateOffsetY()
        return nTransY

func setRotation_degree( nValue)
        n_degree = nValue
        return

func Rotation_degree()
        return n_degree

func setClickEvent( cEvent)
        oButton.setClickEvent(cEvent)
        return

func braceend()
        draw()
        return

func setEnabled(value)
        oButton.setenabled(value)
        return

 func setButtonColor(color)
         colorIt = "background-color:" + color
         oButton.setstylesheet(colorIt)
         return

func draw()
         picture = new qpicture()
         color   = new qcolor() { setrgb(0,0,255,255) }
         pen     = new qpen()   { setcolor(color) setwidth(10) }

         painter = new qpainter()
         {
                       begin(picture)
                       setpen(pen)
                       oFont = new qfont("Courier New",12,75,0)
                       oFont.setpointsize(20)
                       setfont(oFont)
                       if n_degree = 0
                          if btn_degree[empty] [1]="In" p1 = -8 p2=0
                          translate(p1,p2) ok ok
                       if n_degree = 0
                          if btn_degree[empty] [1]<10 p1 = 10 p2=10 else p1=5 p2=10 ok
                          translate(p1,p2)
                       but n_degree = 90
                             if btn_degree[empty] [1]<10 p=-10 else p=-15 ok
                             translate(10,p)
                       but n_degree = 180
                              if btn_degree[empty] [1]<10 p1= 30 p2=-10 else p1=35 p2=-10 ok
                              translate(p1,p2)
                       but n_degree = 270
                             if btn_degree[empty] [1]<10 p=10 else p=15 ok
                             translate(30,p)
                       but n_degree = -90
                             if btn_degree[empty] [1]<10 p=10 else p=15 ok
                             translate(30,p)
                       but n_degree = -180
```

```
                        if btn_degree[empty] [1]<10 p1=30 p2=-10 else p1=35 p2=-10 ok
                        translate(p1,p2)
                    but n_degree = -270
                        if btn_degree[empty] [1]<10 p1=10 p2=-10 else p1=10 p2=-15 ok
                        translate(p1,p2)
                    ok
                    rotate(n_degree)
            drawtext(0,0,this.Text())
                    endpaint()
        }
        oLabel {
                setpicture(picture)
                 show()
                }
        return
```

Output:

CalmoSoft Fifteen Puzzle Game in Ring - video (https://youtu.be/IZqpYctv8Zs)

Ruby

```ruby
require 'io/console'

class Board
  SIZE = 4
  RANGE = 0...SIZE

  def initialize
    width = (SIZE*SIZE-1).to_s.size
    @frame = ("+" + "-"*(width+2)) * SIZE + "+"
    @form = "| %#{width}d " * SIZE + "|"
    @step = 0
    @orign = [*0...SIZE*SIZE].rotate.each_slice(SIZE).to_a.freeze
    @board = @orign.map{|row | row.dup}
    randomize
    draw
    message
    play
  end

  private

  def randomize
    @board[0][0], @board[SIZE-1][SIZE-1] = 0, 1
    @board[SIZE-1][0], @board[0][SIZE-1] = @board[0][SIZE-1], @board[SIZE-1][0]
    x, y, dx, dy = 0, 0, 1, 0
    50.times do
      nx,ny = [[x+dx,y+dy], [x+dy,y-dx], [x-dy,y+dx]]
                .select{|nx,ny| RANGE.include?(nx) and RANGE.include?(ny)}
                .sample
      @board[nx][ny], @board[x][y] = 0, @board[nx][ny]
      x, y, dx, dy = nx, ny, nx-x, ny-y
    end
    @x, @y = x, y
  end

  def draw
    puts "\e[H\e[2J"
    @board.each do |row|
      puts @frame
      puts (@form % row).sub(" 0 ", "   ")
    end
    puts @frame
    puts "Step: #{@step}"
  end

  DIR = {up: [-1,0], down: [1,0], left: [0,-1], right: [0,1]}
  def move(direction)
    dx, dy = DIR[direction]
    nx, ny = @x + dx, @y + dy
    if RANGE.include?(nx) and RANGE.include?(ny)
      @board[nx][ny], @board[@x][@y] = 0, @board[nx][ny]
```

```
      @x, @y = nx, ny
      @step += 1
      draw
    end
  end

  def play
    until @board == @orign
      case  key_in
      when "\e[A", "w" then move(:up)
      when "\e[B", "s" then move(:down)
      when "\e[C", "d" then move(:right)
      when "\e[D", "a" then move(:left)

      when "q","\u0003","\u0004"  then exit
      when "h"  then message
      end
    end

    puts "Congratulations, you have won!"
  end

  def key_in
    input = STDIN.getch
    if input == "\e"
      2.times {input << STDIN.getch}
    end
    input
  end

  def message
    puts <<~EOM
      Use the arrow-keys or WASD on your keyboard to push board in the given direction.
      PRESS q TO QUIT (or Ctrl-C or Ctrl-D)
    EOM
  end
end

Board.new
```

Output:

```
+----+----+----+----+
|  5 |  7 |  2 | 13 |
+----+----+----+----+
|  6 |    |  8 | 12 |
+----+----+----+----+
| 10 |  3 |  1 | 15 |
+----+----+----+----+
|  9 |  4 | 14 | 11 |
+----+----+----+----+
Step: 0
Use the arrow-keys or WASD on your keyboard to push board in the given direction.
PRESS q TO QUIT (or Ctrl-C or Ctrl-D)
```

Library: RubyGems
Library: JRubyArt

Gui Version:-

```
DIM = 100
SOLUTION = %w[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0].freeze

attr_reader :bcolor, :dark, :light, :tcolor, :space, :blank, :list

def settings
  size(400, 400)
end

def setup
  sketch_title 'Fifteen Tile Puzzle'
  labels = SOLUTION.shuffle
```

```
  @list = []
  grid(width, height, DIM, DIM) do |y, x|
    list << Tile.new(Vec2D.new(x, y), labels[list.length])
  end
  @tcolor = color(255, 175, 0)
  @bcolor = color(235, 231, 178)
  @dark = color(206, 141, 0)
  @light = color(255, 214, 126)
  text_size(DIM / 2.7)
  text_align(CENTER)
  no_loop
end

def draw
  list.each(&:draw)
end

def mouse_clicked
  inside = list.find_index { |tile| tile.include?(Vec2D.new(mouse_x, mouse_y)) }
  target = list.find_index { |tile| tile.label == '0' }
  source = list[inside].pos
  dest = list[target].pos
  return unless source.dist(dest) == DIM

  list[target].label = list[inside].label
  list[inside].label = '0'
  redraw
end

# Our Tile Boundary Class
class Boundary
  attr_reader :low, :high

  def initialize(low, high)
    @low = low
    @high = high
  end

  def include?(val)
    return false unless (low.x...high.x).cover? val.x

    return false unless (low.y...high.y).cover? val.y

    true
  end
end

# Holds Tile logic and draw (Processing::Proxy give access to Sketch methods)
class Tile
  include Processing::Proxy
  attr_writer :label
  attr_reader :boundary, :label, :pos

  def initialize(pos, lbl)
    @label = lbl
    @pos = pos
    @boundary = Boundary.new(pos, pos + Vec2D.new(DIM, DIM))
  end

  def draw_empty
    fill(bcolor)
    rect(pos.x + 1, pos.y + 1, DIM - 1, DIM - 1)
  end

  def draw_tile
    rect(pos.x + 1, pos.y + 1, DIM - 1, DIM - 1)
    fill(0) # Black text shadow
    text(label, pos.x + DIM / 2 + 1, pos.y + DIM / 2 + text_ascent / 2)
    fill(255)
    text(label, pos.x + DIM / 2, pos.y + DIM / 2 + text_ascent / 2)
    stroke(dark)
    line(pos.x + DIM - 1, pos.y + 1, pos.x + DIM - 1, pos.y + DIM - 2) # Right side shadow
    line(pos.x + 2, pos.y + DIM, pos.x + DIM - 1, pos.y + DIM - 2) # Bottom side shadow
    stroke(light)
    line(pos.x + 2, pos.y - 1, pos.x + 2, pos.y + DIM - 1) # Left bright
    line(pos.x + 2, pos.y + 1, pos.x + DIM - 1, pos.y + 1) # Upper bright
  end
```

```
  def include?(vec)
    boundary.include?(vec)
  end

  def draw
    no_stroke
    return draw_empty if label == '0'

    fill(tcolor)
    draw_tile
  end
end
```

Run BASIC

```
call SetCSS
' ---- fill 15 squares with 1 to 15
dim sq(16)
for i = 1 to 15: sq(i) = i: next

'----- shuffle the squares
[newGame]
for i = 1 to 100        ' Shuffle the squares
    j   = rnd(0) * 16 + 1
    k   = rnd(0) * 16 + 1
    h   = sq(j)
    sq(j)   = sq(k)
    sq(k)   = h
next i

' ---- show the squares
[loop]
cls
html "<CENTER><TABLE><TR align=center>"
for i = 1 to 16
    html "<TD>"
    if sq(i) <> 0 then
        button  #pick, str$(sq(i)), [pick]
            #pick setkey(str$(i))
            #pick cssclass("lBtn")
    end if
    html "</TD>"
    if i mod 4 = 0 then html "</TR><TR align=center>"
next i
html "</table>"
wait

' ---- Find what square they picked
[pick]
picked  = val(EventKey$)
move    = 0             '               0000000001111111
if picked - 1 > 0 then  ' LEFT          1234567890123456
                if mid$(" *** *** *** ***",picked,1) = "*" and sq(picked -1) = 0 then move = -1 :end if
if picked + 1 < 17 then ' RIGHT
                if mid$("*** *** *** *** ",picked,1) = "*" and sq(picked +1) = 0 then move =  1 :end if
if picked - 4 > 0  then ' UP
                if mid$("    ************",picked,1) = "*" and sq(picked -4) = 0 then move = -4 :end if
if picked + 4 < 17 then ' DOWN
                if mid$("************    ",picked,1) = "*" and sq(picked +4) = 0 then move =  4 :end if
' ---- See if they picked a valid square next to the blank square
if move = 0 then
    print "Invalid move: ";sq(picked)
    wait
end if

' ---- Valid squire, switch it with the blank square
sq(picked + move) = sq(picked) ' move to the empty square
sq(picked) = 0
for i = 1 to 15  ' ---- If they got them all in a row they are a winner
    if sq(i) <> i then goto [loop]
next i

print "----- You are a winner -----"
input "Play again (Y/N)";a$
```

```
if a$ = "Y" then goto [newGame]     ' set up new game
end

' ---- Make the squares look nice
SUB SetCSS
CSSClass ".lBtn", "{
background:wheat;border-width:5px;width:70px;
Text-Align:Center;Font-Size:24pt;Font-Weight:Bold;Font-Family:Arial;
}"
END SUB
```

Output: File:KokengeGame15.jpg

Rust

Library: rand

```
extern crate rand;

use std::collections::HashMap;
use std::fmt;

use rand::Rng;
use rand::seq::SliceRandom;

#[derive(Copy, Clone, PartialEq, Debug)]
enum Cell {
    Card(usize),
    Empty,
}

#[derive(Eq, PartialEq, Hash, Debug)]
enum Direction {
    Up,
    Down,
    Left,
    Right,
}

enum Action {
    Move(Direction),
    Quit,
}

type Board = [Cell; 16];
const EMPTY: Board = [Cell::Empty; 16];

struct P15 {
    board: Board,
}

impl P15 {
    fn new() -> Self {
        let mut board = EMPTY;
        for (i, cell) in board.iter_mut().enumerate().skip(1) {
            *cell = Cell::Card(i);
        }

        let mut rng = rand::thread_rng();

        board.shuffle(&mut rng);
        if !Self::is_valid(board) {
            // random swap
            let i = rng.gen_range(0, 16);
            let mut j = rng.gen_range(0, 16);
            while j == i {
                j = rng.gen_range(0, 16);
            }
            board.swap(i, j);
        }

        Self { board }
    }
```

```rust
    fn is_valid(mut board: Board) -> bool {
        // TODO: optimize
        let mut permutations = 0;

        let pos = board.iter().position(|&cell| cell == Cell::Empty).unwrap();

        if pos != 15 {
            board.swap(pos, 15);
            permutations += 1;
        }

        for i in 1..16 {
            let pos = board
                .iter()
                .position(|&cell| match cell {
                    Cell::Card(value) if value == i => true,
                    _ => false,
                })
                .unwrap();

            if pos + 1 != i {
                board.swap(pos, i - 1);
                permutations += 1;
            }
        }

        permutations % 2 == 0
    }

    fn get_empty_position(&self) -> usize {
        self.board.iter().position(|&c| c == Cell::Empty).unwrap()
    }

    fn get_moves(&self) -> HashMap<Direction, Cell> {
        let mut moves = HashMap::new();
        let i = self.get_empty_position();

        if i > 3 {
            moves.insert(Direction::Up, self.board[i - 4]);
        }
        if i % 4 != 0 {
            moves.insert(Direction::Left, self.board[i - 1]);
        }
        if i < 12 {
            moves.insert(Direction::Down, self.board[i + 4]);
        }
        if i % 4 != 3 {
            moves.insert(Direction::Right, self.board[i + 1]);
        }
        moves
    }

    fn play(&mut self, direction: &Direction) {
        let i = self.get_empty_position();
        // This is safe because `ask_action` only returns legal moves
        match *direction {
            Direction::Up => self.board.swap(i, i - 4),
            Direction::Left => self.board.swap(i, i - 1),
            Direction::Right => self.board.swap(i, i + 1),
            Direction::Down => self.board.swap(i, i + 4),
        };
    }

    fn is_complete(&self) -> bool {
        self.board.iter().enumerate().all(|(i, &cell)| match cell {
            Cell::Card(value) => value == i + 1,
            Cell::Empty => i == 15,
        })
    }
}

impl fmt::Display for P15 {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        r#try!(write!(f, "+----+----+----+----+\n"));
        for (i, &cell) in self.board.iter().enumerate() {
            match cell {
                Cell::Card(value) => r#try!(write!(f, "| {:2} ", value)),
                Cell::Empty => r#try!(write!(f, "|    ")),
```

```
            }

            if i % 4 == 3 {
                r#try!(write!(f, "|\n"));
                r#try!(write!(f, "+----+----+----+----+\n"));
            }
        }
        Ok(())
    }
}

fn main() {
    let mut p15 = P15::new();

    for turns in 1.. {
        println!("{}", p15);
        match ask_action(&p15.get_moves()) {
            Action::Move(direction) => {
                p15.play(&direction);
            }
            Action::Quit => {
                println!("Bye !");
                break;
            }
        }

        if p15.is_complete() {
            println!("Well done ! You won in {} turns", turns);
            break;
        }
    }
}

fn ask_action(moves: &HashMap<Direction, Cell>) -> Action {
    use std::io::{self, Write};
    use Action::*;
    use Direction::*;

    println!("Possible moves:");

    if let Some(&Cell::Card(value)) = moves.get(&Up) {
        println!("\tU) {}", value);
    }
    if let Some(&Cell::Card(value)) = moves.get(&Left) {
        println!("\tL) {}", value);
    }
    if let Some(&Cell::Card(value)) = moves.get(&Right) {
        println!("\tR) {}", value);
    }
    if let Some(&Cell::Card(value)) = moves.get(&Down) {
        println!("\tD) {}", value);
    }
    println!("\tQ) Quit");
    print!("Choose your move : ");
    io::stdout().flush().unwrap();

    let mut action = String::new();
    io::stdin().read_line(&mut action).expect("read error");
    match action.to_uppercase().trim() {
        "U" if moves.contains_key(&Up) => Move(Up),
        "L" if moves.contains_key(&Left) => Move(Left),
        "R" if moves.contains_key(&Right) => Move(Right),
        "D" if moves.contains_key(&Down) => Move(Down),
        "Q" => Quit,
        _ => {
            println!("Unknown action: {}", action);
            ask_action(moves)
        }
    }
}
```

Scala

```
import java.util.Random
```

```scala
import jline.console._

import scala.annotation.tailrec
import scala.collection.immutable
import scala.collection.parallel.immutable.ParVector

object FifteenPuzzle {
  def main(args: Array[String]): Unit = play()

  @tailrec def play(len: Int = 1000): Unit = if(gameLoop(Board.randState(len))) play(len)
  def gameLoop(board: Board): Boolean = {
    val read = new ConsoleReader()
    val km = KeyMap.keyMaps().get("vi-insert")
    val opMap = immutable.HashMap[Operation, Char](
      Operation.PREVIOUS_HISTORY -> 'u',
      Operation.BACKWARD_CHAR -> 'l',
      Operation.NEXT_HISTORY -> 'd',
      Operation.FORWARD_CHAR -> 'r')

    @tailrec
    def gloop(b: Board): Boolean = {
      println(s"\u001B[2J\u001B[2;0H$b\n←↑→↓q")
      if(b.isSolved) println("Solved!\nPlay again? (y/n)")

      read.readBinding(km) match{
        case Operation.SELF_INSERT => read.getLastBinding match{
          case "q" => false
          case "y" if b.isSolved => true
          case "n" if b.isSolved => false
          case _ => gloop(b)
        }
        case op: Operation if opMap.isDefinedAt(op) => gloop(b.move(opMap(op)))
        case _ => gloop(b)
      }
    }

    gloop(board)
  }

  case class Board(mat: immutable.HashMap[(Int, Int), Int], x: Int, y: Int) {
    def move(mvs: Seq[Char]): Board = mvs.foldLeft(this){case (b, m) => b.move(m)}
    def move(mov: Char): Board = mov match {
      case 'r' if x < 3 => Board(mat ++ Seq(((x, y), mat((x + 1, y))), ((x + 1, y), 0)), x + 1, y)
      case 'l' if x > 0 => Board(mat ++ Seq(((x, y), mat((x - 1, y))), ((x - 1, y), 0)), x - 1, y)
      case 'd' if y < 3 => Board(mat ++ Seq(((x, y), mat((x, y + 1))), ((x, y + 1), 0)), x, y + 1)
      case 'u' if y > 0 => Board(mat ++ Seq(((x, y), mat((x, y - 1))), ((x, y - 1), 0)), x, y - 1)
      case _ => this
    }

    def isSolved: Boolean = sumDist == 0
    def sumDist: Int = mat.to(LazyList).map{ case ((a, b), n) => if(n == 0) 6 - a - b else (a + b - ((n - 1) % 4)
- ((n - 1) / 4)).abs }.sum

    override def toString: String = {
      val lst = mat.toVector.map { case ((a, b), n) => (4 * b + a, n) }.sortWith(_._1 < _._1).map(_._2)
      lst.map { n => if (n == 0) "  " else f"$n%2d" }.grouped(4).map(_.mkString(" ")).mkString("\n")
    }
  }

  object Board {
    val moves: Vector[Char] = Vector('r', 'l', 'd', 'u')

    def apply(config: Vector[Int]): Board = {
      val ind = config.indexOf(0)
      val formed = config.zipWithIndex.map { case (n, i) => ((i % 4, i / 4), n) }
      val builder = immutable.HashMap.newBuilder[(Int, Int), Int]
      builder ++= formed
      Board(builder.result, ind % 4, ind / 4)
    }

    def solveState: Board = apply((1 to 15).toVector :+ 0)
    def randState(len: Int, rand: Random = new Random()): Board = Iterator
      .fill(len)(moves(rand.nextInt(4)))
      .foldLeft(Board.solveState) { case (state, mv) => state.move(mv) }
  }
}
```

Scheme

```
(import (scheme base)
        (scheme read)
        (scheme write)
        (srfi 27))    ; random numbers

(define *start-position* #(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 #\space))
(random-source-randomize! default-random-source)

;; return a 16-place vector with the tiles randomly shuffled
(define (create-start-position)
  (let ((board (vector-copy *start-position*)))
    (do ((i 0 (+ 1 i))
         (moves (find-moves board) (find-moves board)))
      ((and (>= i 100)
            (not (finished? board)))
       board)
      (make-move board
                 (list-ref moves (random-integer (length moves)))))))

;; return index of space
(define (find-space board)
  (do ((i 0 (+ 1 i)))
    ((equal? #\space (vector-ref board i)) i)))

;; return a list of symbols indicating available moves
(define (find-moves board)
  (let* ((posn (find-space board))
         (row (quotient posn 4))
         (col (remainder posn 4))
         (result '()))
    (when (> row 0) (set! result (cons 'up result)))
    (when (< row 3) (set! result (cons 'down result)))
    (when (> col 0) (set! result (cons 'left result)))
    (when (< col 3) (set! result (cons 'right result)))
    result))

;; make given move - assume it is legal
(define (make-move board move)
  (define (swap posn-1 posn-2)
    (let ((tmp (vector-ref board posn-1)))
      (vector-set! board posn-1 (vector-ref board posn-2))
      (vector-set! board posn-2 tmp)))
  ;
  (let ((posn (find-space board)))
    (case move
      ((left)
       (swap posn (- posn 1)))
      ((right)
       (swap posn (+ posn 1)))
      ((up)
       (swap posn (- posn 4)))
      ((down)
       (swap posn (+ posn 4))))))

(define (finished? board)
  (equal? board *start-position*))

(define (display-board board)
  (do ((i 0 (+ 1 i)))
    ((= i (vector-length board)) (newline))
    (when (zero? (modulo i 4)) (newline))
    (let ((curr (vector-ref board i)))
      (display curr)
      (display (if (and (number? curr)
                        (> curr 9))
                   " "
                   "  ")))))

;; the main game loop
(define (play-game)
  (let ((board (create-start-position)))
    (do ((count 1 (+ count 1))
         (moves (find-moves board) (find-moves board)))
      ((finished? board)
```

```
       (display (string-append "\nCOMPLETED PUZZLE in "
                               (number->string count)
                               " moves\n")))
      (display-board board)
      (display "Enter a move: ") (display moves) (newline)
      (let ((move (read)))
        (if (memq move moves)
          (make-move board move)
          (display "Invalid move - try again"))))))

(play-game)
```

Output:

```
1  2     3
5  7  6  11
13 14 8  4
10 9  15 12
Enter a move: (right left down)
right

1  2  3
5  7  6  11
13 14 8  4
10 9  15 12
Enter a move: (left down)
down

1  2  3  11
5  7  6
13 14 8  4
10 9  15 12
Enter a move: (left down up)
down

1  2  3  11
5  7  6  4
13 14 8
10 9  15 12
Enter a move: (left down up)
```

Scilab

```
tiles=[1:15,0];
solution=[tiles(1:4);...
          tiles(5:8);...
          tiles(9:12);...
          tiles(13:16)];
solution=string(solution);
solution(16)=" ";

init_pos=grand(1,"prm",tiles);
puzzle=[init_pos(1:4);...
        init_pos(5:8);...
        init_pos(9:12);...
        init_pos(13:16)];
puzzle=string(puzzle);

blank_pos=[];
for i=1:4
    for j=1:4
        if puzzle(i,j)=="0" then
            blank_pos=[i,j];
        end
    end
end

clear i j

puzzle(blank_pos(1),blank_pos(2))=" ";

n_moves=0;
```

```
solved=%F;
while ~solved
    disp(puzzle); mprintf("\n");

    neighbours=[0 -1;...
                -1  0;...
                 0 +1;...
                +1  0];
    neighbours(:,1)=neighbours(:,1)+blank_pos(1);
    neighbours(:,2)=neighbours(:,2)+blank_pos(2);
    neighbours=[neighbours zeros(4,1)]

    i=0;
    for i=1:4
        if ~(neighbours(i,1)<1 | neighbours(i,1)>4 |...
           neighbours(i,2)<1 | neighbours(i,2)>4) then
           neighbours(i,3)=evstr(puzzle(neighbours(i,1),neighbours(i,2)));
       end
    end

    valid_move=%F;
    while ~valid_move
        move_tile=[];
        move_tile=input("Enter tile you want to move (0 to exit):");
        if sum(move_tile==neighbours(:,3)) & move_tile~=0 then
            valid_move=%T;
            n_moves=n_moves+1;
        elseif move_tile==0 then
            disp("Exit");
            abort
        else
            disp(puzzle);
            disp("Invalid input");
        end
    end

    neighb_i=find(neighbours(:,3)'==move_tile);
    puzzle(neighbours(neighb_i,1),neighbours(neighb_i,2))=" ";
    puzzle(blank_pos(1),blank_pos(2))=string(move_tile);
    blank_pos=neighbours(neighb_i,1:2);

    if sum(puzzle==solution)==16 then
        solved=%T;
        disp(puzzle);
        mprintf("\n"+...
                "   _____       _                 _ _ \n"+...
                "  / ____|     | |               | | |\n"+...
                " | (___   ___ | |_   _____  __| | |\n"+...
                "  \\___ \\ / _ \\| \\ \\ / / _ \\/ _` | |\n"+...
                "  ____) | (_) | |\\ V /  __/ (_| |_|\n"+...
                " |_____/ \\___/|_| \\_/ \\___|\\__,_(_)\n")
    end
end

disp("Solved in "+string(n_moves)+" moves.");
```

Output:

This test was run while making `init_pos=[1 2 3 4 5 6 7 8 9 14 10 11 13 0 15 12]`.

```
!1   2   3   4   !
!               !
!5   6   7   8   !
!               !
!9   14  10  11  !
!               !
!13      15  12  !

Enter tile you want to move (0 to exit):
14

!1   2   3   4   !
!               !
!5   6   7   8   !
!               !
```

```
!9       10  11  !
!                !
!13  14  15  12  !

Enter tile you want to move (0 to exit):
10


!1   2   3   4   !
!                !
!5   6   7   8   !
!                !
!9   10      11  !
!                !
!13  14  15  12  !

Enter tile you want to move (0 to exit):
11


!1   2   3   4   !
!                !
!5   6   7   8   !
!                !
!9   10  11      !
!                !
!13  14  15  12  !

Enter tile you want to move (0 to exit):
12


!1   2   3   4   !
!                !
!5   6   7   8   !
!                !
!9   10  11  12  !
!                !
!13  14  15      !

   _____      _                 _ _
  / ____|    | |               | | |
 | (___   ___ | |_   _____  __| | |
  \___ \ / _ \| \ \ / / _ \/ _` | |
  ____) | (_) | |\ V /  __/ (_| |_|
 |_____/ \___/|_| \_/ \___|\__,_(_)

 Solved in 4 moves.
```

Simula

```
BEGIN
    CLASS FIFTEENPUZZLE(NUMTILES, SIDE, WIDTH, SEED);
        INTEGER NUMTILES, SIDE, WIDTH, SEED;
    BEGIN
        INTEGER ARRAY TILES(0:NUMTILES);
        INTEGER BLANKPOS;

        PROCEDURE INVARIANT;
        BEGIN
            INTEGER ARRAY UQ(0:NUMTILES);
            INTEGER I;
            FOR I := 0 STEP 1 UNTIL NUMTILES DO UQ(I) := -1;
            FOR I := 0 STEP 1 UNTIL NUMTILES DO
            BEGIN
                INTEGER T;
                T := TILES(I);
                IF UQ(T) <> -1 THEN ERROR("TILES ARE NOT UNIQUE");
                UQ(T) := T;
            END;
            IF TILES(BLANKPOS) <> 0 THEN ERROR("BLANKPOS IS NOT BLANK");
        END;

        PROCEDURE SHUFFLE;
```

```
BEGIN
    BOOLEAN B;
    WHILE NOT B DO
    BEGIN
        INTEGER N;
        RESET;
        ! DON'T INCLUDE THE BLANK SPACE IN THE SHUFFLE, LEAVE IT
        ! IN THE HOME POSITION ;
        N := NUMTILES;
        WHILE N > 1 DO
        BEGIN
            INTEGER R, TMP;
            R := UNIFORM(0, N, SEED); N := N - 1;
            TMP := TILES(R);
            TILES(R) := TILES(N);
            TILES(N) := TMP;
        END;
        B := ISSOLVABLE;
    END;
    INVARIANT;
END;

PROCEDURE RESET;
BEGIN
    INTEGER I;
    FOR I := 0 STEP 1 UNTIL NUMTILES DO
        TILES(I) := MOD((I + 1), NUMTILES + 1);
    BLANKPOS := NUMTILES;
    INVARIANT;
END;

! ONLY HALF THE PERMUTATIONS OF THE PUZZLE ARE SOLVABLE.
! WHENEVER A TILE IS PRECEDED BY A TILE WITH HIGHER VALUE IT COUNTS
! AS AN INVERSION. IN OUR CASE, WITH THE BLANK SPACE IN THE HOME
! POSITION, THE NUMBER OF INVERSIONS MUST BE EVEN FOR THE PUZZLE
! TO BE SOLVABLE.
! SEE ALSO:
! WWW.CS.BHAM.AC.UK/~MDR/TEACHING/MODULES04/JAVA2/TILESSOLVABILITY.HTML
;

BOOLEAN PROCEDURE ISSOLVABLE;
BEGIN
    INTEGER COUNTINVERSIONS;
    INTEGER I, J;
    FOR I := 0 STEP 1 UNTIL NUMTILES - 1 DO
        FOR J := 0 STEP 1 UNTIL I - 1 DO
            IF TILES(J) > TILES(I) THEN
                COUNTINVERSIONS := COUNTINVERSIONS + 1;
    ISSOLVABLE := MOD(COUNTINVERSIONS, 2) = 0;
END;

PROCEDURE PRINTBOARD;
BEGIN
    INTEGER I, J;

    PROCEDURE PRINTLINE;
    BEGIN
      INTEGER ROW, COL;
      !      +-----+-----+-----+-----+ ;
      FOR ROW := 1 STEP 1 UNTIL SIDE DO
      BEGIN
          OUTCHAR('+');
          FOR COL := 0 STEP 1 UNTIL WIDTH DO OUTCHAR('-');
      END;
      OUTCHAR('+');
      OUTIMAGE;
    END;

    PROCEDURE PRINTCELL(T); INTEGER T;
    BEGIN
        IF T = 0 THEN
        BEGIN
            INTEGER R;
            FOR R := 1 STEP 1 UNTIL WIDTH DO
                OUTCHAR(' ');
        END
        ELSE OUTINT(T, WIDTH);
        OUTCHAR(' ');
```

```
    END;

    !      +-----+-----+-----+-----+
    !      |  1  |  2  |  3  |  4  |
    !      +-----+-----+-----+-----+
    !      |  5  |  6  |  7  |  8  |
    !      +-----+-----+-----+-----+
    !      |  9  |  10 |  11 |     |
    !      +-----+-----+-----+-----+
    !      |  13 |  14 |  15 |  12 |
    !      +-----+-----+-----+-----+   ;

    FOR I := 1 STEP 1 UNTIL SIDE DO
    BEGIN
        PRINTLINE;
        OUTCHAR('|');
        FOR J := 1 STEP 1 UNTIL SIDE DO
        BEGIN
            INTEGER T;
            T := TILES((I - 1) * SIDE + (J - 1));
            PRINTCELL(T);
            OUTCHAR('|');
        END;
        OUTIMAGE;
    END;
    PRINTLINE;
END;

BOOLEAN PROCEDURE DONE;
BEGIN
    BOOLEAN ORDERED;
    INTEGER I, EXPECT;
    ORDERED := TRUE;
    EXPECT := 1;
    FOR I := 0 STEP 1 UNTIL NUMTILES - 1 DO
    BEGIN
        IF I <> BLANKPOS THEN
        BEGIN
            IF TILES(I) <> EXPECT THEN
                ORDERED := FALSE;
            EXPECT := EXPECT + 1;
        END;
    END;
    DONE := ORDERED;
END;

PROCEDURE REQUEST;
BEGIN
    INTEGER ARRAY CANDIDATES(1:4);
    INTEGER I, CANDCOUNT, CHOOSE;
    BOOLEAN VALIDINPUT;

    PROCEDURE ADDCAND(IDX); INTEGER IDX;
    BEGIN
        IF IDX >= 0 AND IDX <= NUMTILES THEN
        BEGIN
            CANDCOUNT := CANDCOUNT + 1;
            CANDIDATES(CANDCOUNT) := TILES(IDX);
        END;
    END;

    PRINTBOARD;

    IF BLANKPOS <= NUMTILES - SIDE     THEN ADDCAND(BLANKPOS + SIDE);
    IF BLANKPOS >= SIDE                THEN ADDCAND(BLANKPOS - SIDE);
    IF MOD(BLANKPOS, SIDE) <> SIDE - 1 THEN ADDCAND(BLANKPOS + 1);
    IF MOD(BLANKPOS, SIDE) <> 0        THEN ADDCAND(BLANKPOS - 1);

    WHILE NOT VALIDINPUT DO
    BEGIN
        OUTTEXT("YOUR MOVE: ");

        FOR I := 1 STEP 1 UNTIL CANDCOUNT DO
            OUTINT(CANDIDATES(I), SIDE);
        OUTIMAGE;
        CHOOSE := ININT;

        FOR I := 1 STEP 1 UNTIL CANDCOUNT DO
```

```
                    IF CHOOSE = CANDIDATES(I) THEN
                    BEGIN
                        INTEGER LOOKUP;
                        FOR LOOKUP := 0 STEP 1 UNTIL NUMTILES DO
                            IF NOT VALIDINPUT AND TILES(LOOKUP) = CHOOSE THEN
                            BEGIN
                                TILES(BLANKPOS) := TILES(LOOKUP);
                                TILES(LOOKUP) := 0;
                                BLANKPOS := LOOKUP;
                                INVARIANT;
                                VALIDINPUT := TRUE;
                            END;
                    END;

                IF NOT VALIDINPUT THEN
                BEGIN
                    OUTTEXT("INVALID INPUT!");
                    OUTIMAGE;
                END;
            END;
        END;

        SHUFFLE;
    END;

    REF(FIFTEENPUZZLE) P;

    OUTTEXT("INPUT RANDOM SEED: ");
    OUTIMAGE;
    P :- NEW FIFTEENPUZZLE(15, 4, 3, ININT); ! ININT = RANDOM SEED ;
    WHILE NOT P.DONE DO
        P.REQUEST;
    P.PRINTBOARD;
END.
```

Output:

```
INPUT RANDOM SEED:
456
+----+----+----+----+
| 15 |  8 | 13 | 14 |
+----+----+----+----+
|  1 |  2 |  9 |  4 |
+----+----+----+----+
| 12 |  3 |  5 |  7 |
+----+----+----+----+
| 11 |  6 | 10 |    |
+----+----+----+----+
YOUR MOVE:    7  10
7
+----+----+----+----+
| 15 |  8 | 13 | 14 |
+----+----+----+----+
|  1 |  2 |  9 |  4 |
+----+----+----+----+
| 12 |  3 |  5 |    |
+----+----+----+----+
| 11 |  6 | 10 |  7 |
+----+----+----+----+
YOUR MOVE:    7   4   5
4
+----+----+----+----+
| 15 |  8 | 13 | 14 |
+----+----+----+----+
|  1 |  2 |  9 |    |
+----+----+----+----+
| 12 |  3 |  5 |  4 |
+----+----+----+----+
| 11 |  6 | 10 |  7 |
+----+----+----+----+
```

```
YOUR MOVE:    4  14   9
...
```

Standard ML

Works with: SML/NJ
Works with: Moscow ML

```
(* Load required Modules for Moscow ML *)
load "Int";
load "Random";


(* Mutable Matrix *)
signature MATRIX =
sig
    type 'a matrix
    val construct : 'a -> int * int -> 'a matrix
    val size : 'a matrix -> int * int
    val get : 'a matrix -> int * int -> 'a
    val set : 'a matrix -> int * int -> 'a -> unit
end

structure Matrix :> MATRIX =
struct
    (* Array of rows, where the rows are a array of 'a *)
    type 'a matrix = 'a Array.array Array.array

    fun 'a construct (a : 'a) (width, height) : 'a matrix =
        if width < 1 orelse height < 1
            then raise Subscript
            else Array.tabulate (height, fn _ => Array.tabulate (width, fn _ => a))

    fun size b =
        let
            val firstrow = Array.sub (b, 0)
        in
            (Array.length firstrow, Array.length b)
        end


    fun get b (x, y) = Array.sub (Array.sub (b, y), x)

    fun set b (x, y) v = Array.update (Array.sub (b, y), x, v)
end

signature P15BOARD =
sig
    type board
    datatype direction = North | East | South | West

    val construct : int * int -> board
    val emptyField : board -> int * int
    val get : board -> int * int -> int option
    val size : board -> int * int

    exception IllegalMove
    val moves : board -> int list
    val move : board -> int -> unit

    val issolved : board -> bool
end

(* Game Logic and data *)

structure Board :> P15BOARD =
struct
    (* Matrix + empty Field position *)
    type board = int option Matrix.matrix * (int * int) ref

    datatype direction = North | East | South | West

    exception IllegalMove
```

```
    fun numberat width (x, y) = (y*width + x + 1)

    fun construct (width, height) =
        let
            val emptyBoard : int option Matrix.matrix = Matrix.construct NONE (width, height)
        in
            (* Fill the board with numbers *)
            List.tabulate (height, fn y => List.tabulate (width, fn x =>
                    Matrix.set emptyBoard (x, y) (SOME (numberat width (x, y)))));
            (* Clear the last field *)
            Matrix.set emptyBoard (width-1, height-1) NONE;
            (* Return the board *)
            (emptyBoard, ref (width-1, height-1))
        end

    fun emptyField (_, rfield) = !rfield

    fun get (mat, _) (x, y) = Matrix.get mat (x, y)

    fun size (mat, _) = Matrix.size mat

    (* toggle the empty field with a given field *)
    fun toggle (mat, rpos) pos =
        let
            val pos' = !rpos
            val value = Matrix.get mat pos
        in
            Matrix.set mat pos NONE;
            Matrix.set mat pos' value;
            rpos := pos
        end

    (* Get list of positions of the neighbors of a given field *)
    fun neighbors mat (x, y) : (int * int) list =
        let
            val (width, height) = Matrix.size mat
            val directions = [(x, y-1), (x+1, y), (x, y+1), (x-1, y)]
        in
            List.mapPartial (fn pos => SOME (Matrix.get mat pos; pos) handle Subscript => NONE) directions
        end

    fun moves (mat, rpos) =
        let
            val neighbors = neighbors mat (!rpos)
        in
            map (fn pos => valOf (Matrix.get mat pos)) neighbors
        end

    fun move (mat, rpos) m =
        let
            val (hx, hy) = !rpos
            val neighbors = neighbors mat (hx, hy)
            val optNeighbor = List.find (fn pos => SOME m = Matrix.get mat pos) neighbors
        in
            if isSome optNeighbor
            then
                toggle (mat, rpos) (valOf optNeighbor)
            else
                raise IllegalMove
        end

    fun issolved board =
        let
            val (width, height) = size board
            val xs = List.tabulate (width,  fn x => x)
            val ys = List.tabulate (height, fn y => y)
        in
            List.all (fn x => List.all (fn y => (x + 1 = width andalso y + 1 = height) orelse get board (x, y) =
SOME (numberat width (x,y))) ys) xs
        end
end

(* Board Shuffle *)
signature BOARDSHUFFLE =
sig
    val shuffle : Board.board -> int -> unit
end
```

```
structure Shuffle :> BOARDSHUFFLE =
struct
    (*
     * Note: Random Number Interfaces are different in SML/NJ and Moscow ML. Comment out the corresponding
version:
     *)

    (*
    (* SML/NJ - Version *)
    val time = Time.now ()
    val timeInf = Time.toMicroseconds time
    val timens = Int.fromLarge (LargeInt.mod (timeInf, 1073741823));
    val rand = Random.rand (timens, timens)

    fun next n = Random.randRange (0, n) rand
    *)

    (* Moscow ML - Version *)
    val generator = Random.newgen ()
    fun next n = Random.range (0, n) generator


    fun shuffle board 0 = if (Board.issolved board) then shuffle board 1 else ()
      | shuffle board n =
        let
            val moves = Board.moves board
            val move  = List.nth (moves, next (List.length moves - 1))
        in
            Board.move board move;
            shuffle board (n-1)
        end
end


(* Console interface *)

signature CONSOLEINTERFACE =
sig
    val start : unit -> unit
    val printBoard : Board.board -> unit
end

structure Console :> CONSOLEINTERFACE =
struct
    fun cls () = print "\^[[1;1H\^[[2J"

    fun promptNumber prompt =
        let
            val () = print prompt
            (* Input + "\n" *)
            val line   = valOf (TextIO.inputLine TextIO.stdIn)
            val length = String.size line
            val input  = String.substring (line, 0, length - 1)
            val optnum = Int.fromString input
        in
            if isSome optnum
                then valOf optnum
                else (print "Input is not a number.\n"; promptNumber prompt)
        end

    fun fieldToString (SOME x) = Int.toString x
      | fieldToString (NONE  ) = ""

    fun boardToString board =
        let
            val (width, height) = Board.size board
            val xs = List.tabulate (width,  fn x => x)
            val ys = List.tabulate (height, fn y => y)
        in
            foldl (fn (y, str) => (foldl (fn (x, str') => str' ^ (fieldToString (Board.get board (x, y))) ^ "\t")
str xs) ^ "\n") "" ys
        end

    fun printBoard board = print (boardToString board)


    fun loop board =
        let
```

```
            val rvalidInput = ref false
            val rinput      = ref 42
            val () = cls ()
            val () = printBoard board
        in
            (* Ask for a move and repeat until it is a valid move *)
            while (not (!rvalidInput)) do
                (
                    rinput := promptNumber "Input the number to move: ";
                    Board.move board (!rinput);
                    rvalidInput := true
                ) handle Board.IllegalMove => print "Illegal move!\n"
        end


    fun start () =
        let
            val () = cls ()
            val () = print "Welcome to nxm-Puzzle!\n"
            val (width, height) = (promptNumber "Enter the width: ", promptNumber "Enter the height: ")
            val diff = (promptNumber "Enter the difficulty (number of shuffles): ")
            val board = Board.construct (width, height)
        in
            Shuffle.shuffle board diff;
            while (not (Board.issolved board)) do loop board;
            print "Solved!\n"
        end
end


val () = Console.start()
```

Note: The interface for generating random numbers is different in SML/NJ and Moscow ML. Comment out the corresponding parts of code.
The dimensions of the board (eg. 4x4) and the number of shuffles should be entered first.

Tcl

Library: Tk

Works with Tcl/Tk 8.5

This program uses Tk, the graphical counterpart to Tcl. The tiles are made of a grid of buttons, and the text on the buttons is moved around.

The button "New game" selects one of the canned puzzles. The window-title is used to show messages.

```
# 15puzzle_21.tcl - HaJo Gurt -  2016-02-16
# http://wiki.tcl.tk/14403

#: 15-Puzzle - with grid, buttons and colors

 package require Tk

 set progVersion "15-Puzzle v0.21";        # 2016-02-20

 global Msg Moves PuzzNr GoalNr
 set Msg    " "
 set Moves  -1
 set PuzzNr  0
 set GoalNr  0

 set Keys   { 11 12 13 14  21 22 23 24  31 32 33 34  41 42 43 44 }

 set Puzz_T {  T  h  e  F   i  f  t  e   e  n  P  u   z  z  l  e }; # Title
 set Goal_T {  x  x  x  F   i  f  t  e   e  n  x  x   x  x  x  x }; # Title-highlight

 set Puzz_0 {  E  G  P  N   C  A  F  B   D  L  H  I   O  K  M  _ }; # -  / 116
 set Puzz_1 {  C  A  F  B   E  G  P  N   D  L  H  I   O  K  M  _ }; # E  / 156 from Tk-demo
```

```
  set Puzz_2 {  E  O  N  K   M  I  _  G   B  H  L  P   C  F  A  D }; # L  / 139
  set Puzz_3 {  P  G  M  _   E  L  N  D   O  K  H  I   B  C  F  A }; # EK / 146

  set Goal_0 {  A  B  C  D   E  F  G  H   I  K  L  M   N  O  P  _ }; # Rows LTR   / 1:E : 108
  set Goal_1 {  A  E  I  N   B  F  K  O   C  G  L  P   D  H  M  _ }; # Cols forw. / 1:M : 114

  set Puzz $Puzz_T
  set Goal $Goal_T

#---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---

  proc Move {k} {
  # find the key with the empty tile:
    set e -1
    foreach p $::Keys  {
      set t [.key$p cget -text]
      if { $t eq "_" } { set e $p }
    }
    if {$e  <  0} {return 0};   # no key with empty tile found
    if {$k == $e} {return 0};   # click was on the empty tile

    set t [.key$k cget -text]
    .key$e config -text $t
    .key$k config -text "_";
    return 1
  }

  proc Check {} {
    set ok 0
    set i  0
    foreach k $::Keys {
      set t [.key$k cget -text]
      set g [lindex $::Goal $i]
      incr i

      .key$k config -background white
      if { $t eq $g  } { .key$k config -background lightgreen; incr ok }
      if { $t eq "_" } { .key$k config -background gray }
    }

    # Solved:
    update
    if { $ok > 15 && $::Moves > 0} {
      foreach k $::Keys  {
        .key$k flash; bell;
      }
    }
  }

  proc Click {k} {
    set ::Msg ""
    set val [.key$k cget -text]
    set ok [Move $k]

    incr ::Moves $ok
    wm title . "$::Moves moves"
    Check
  }

  proc ShowKeys {} {
    set i 0
    foreach k $::Keys  {
      set t [lindex $::Puzz $i]
      incr i
      .key$k config -text $t -background white;
    }
    Check
  }

  proc NewGame {N} {
    global Msg Moves PuzzNr GoalNr

    incr  PuzzNr $N
    if { $PuzzNr > 3} { set PuzzNr 0 }

                          set ::Goal $::Goal_0;
    if { $GoalNr == 1} { set ::Goal $::Goal_1; }
```

```
    if { $PuzzNr == 0} { set ::Puzz $::Puzz_0; }
    if { $PuzzNr == 1} { set ::Puzz $::Puzz_1; }
    if { $PuzzNr == 2} { set ::Puzz $::Puzz_2; }
    if { $PuzzNr == 3} { set ::Puzz $::Puzz_3; }

                  set Msg "Try again"
    if { $N>0 } { set Msg "New game" }

    set Moves 0
    ShowKeys
    wm title . "$Msg "
  }

#---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---

  button .reset   -text "Restart"  -fg blue -command {NewGame  0}
  button .newGame -text "New Game" -fg red  -command {NewGame +1}

  foreach k $::Keys {
    button .key$k -text "$k" -width 4 -command "Click $k"
  }

  grid .newGame x .reset x -sticky nsew

  grid .key11 .key12 .key13 .key14  -sticky nsew  -padx 2 -pady 2
  grid .key21 .key22 .key23 .key24  -sticky nsew  -padx 2 -pady 2
  grid .key31 .key32 .key33 .key34  -sticky nsew  -padx 2 -pady 2
  grid .key41 .key42 .key43 .key44  -sticky nsew  -padx 2 -pady 2

  grid configure .newGame .reset  -columnspan 2 -padx 4

  ShowKeys; Check
  wm title . $progVersion
  focus -force .
  wm resizable . 0 0

# For some more versions, see:  http://wiki.tcl.tk/15067 : Classic 15 Puzzle and http://wiki.tcl.tk/15085 : N-Puzzle
```

UNIX Shell

Works with: Bourne Again SHell version 4+

This plays the game in an ANSI-compliant terminal using vi/nethack movement keys to slide the tiles.

```
#!/usr/bin/env bash
main() {
  local puzzle=({1..15} " ") blank moves_kv i key count total last
  printf '\n'
  show_puzzle "${puzzle[@]}"
  printf '\nPress return to scramble.'
  read _
  IFS= readarray -t puzzle < <(scramble "${puzzle[@]}")
  printf '\r\e[A\e[A\e[A\e[A\e[A\e[A'
  show_puzzle "${puzzle[@]}"
  printf '\nUse hjkl to slide tiles. '
  printf '\r\e[A\e[A\e[A\e[A\e[A'
  total=0
  while ! solved "${puzzle[@]}"; do
    show_puzzle "${puzzle[@]}"
    { read blank; readarray -t moves_kv; } < <(find_moves "${puzzle[@]}")
    local count=${#moves_kv[@]}/2
    local -A moves
    for (( i=0; i<count; i++ )); do
      moves[${moves_kv[i]}]=${moves_kv[i+count]}
    done
    read -r -n 1 key
    printf '\r '
    if [[ "$key" = u ]]; then
      (( total -= 2 ))
      case "$last" in
        h) key=l;;
```

```
        j) key=k;;
        k) key=j;;
        l) key=h;;
      esac
    fi
    if [[ -n "${moves[$key]}" ]]; then
      last=$key
      (( total++ ))
      i=${moves[$key]}
      puzzle[$blank]=${puzzle[i]}
      puzzle[i]=' '
    fi
    printf '\r\e[A\e[A\e[A\e[A'
  done
  show_puzzle "${puzzle[@]}"
  printf '\nSolved in %d moves.     \n' "$total"
}


solved() {
  local solved=({1..15} ' ')
  [[ "${puzzle[*]}" == "${solved[*]}" ]]
}

show_puzzle() {
  printf '%2s %2s %2s %2s\n' "$@"
}

find_moves() {
  local puzzle=("$@")
  local i j blank
  for (( i=0; i<${#puzzle[@]}; ++i )); do
    if [[ "${puzzle[i]}" == " " ]]; then
      blank=$i
      break
    fi
  done
  local -A moves=()
  if (( blank%4 )); then
    # swapping blank with tile to its left
    # is sliding that tile right, which is the l key
    moves['l']=$(( blank-1 ))
  fi
  if (( blank%4 != 3 )); then
    moves['h']=$(( blank+1 )) # left
  fi
  if (( blank >= 4 )); then
    moves['j']=$(( blank-4 )) # down
  fi
  if (( blank < 12 )); then
    moves['k']=$(( blank+4 )) # up
  fi
  printf '%s\n' "$blank" "${!moves[@]}" "${moves[@]}"
}

scramble() {
  local puzzle=("$@") i j
  for (( i=0; i<256; ++i )); do
    local blank moves
    { read blank; readarray -t moves; } < <(find_moves "${puzzle[@]}")
    moves=(${moves[@]:${#moves[@]}/2})
    local dir=$(( RANDOM % ${#moves[@]} ))
    j=${moves[dir]}
    puzzle[blank]=${puzzle[j]}
    puzzle[j]=' '
  done
  printf '%s\n' "${puzzle[@]}"
}

main "$@"
```

Output:

```
 1  2  3  4
 5  6  7  8
```

```
 9 10 11 12
13 14 15

Press return to scramble.
```

VBA

This allows the user to specify the size of the grid (N x N). Only solvable layouts are displayed.

This uses InputBoxes to display the grid of tiles, and prompts the user to enter the number on the tile they wish to move into the empty space.

The last move is displayed on the input box, or an error message if an invalid move is attempted. When the puzzle is solved, the move count is displayed.

```
Public iSide As Integer
Public iSize As Integer
Public iGrid() As Integer
Public lMoves As Long
Public sMessage As String
Public Const sTitle As String = "Tile Puzzle"


Sub PlayGame()
    Dim iNum As Integer
    Dim i As Integer
    Dim vInput As Variant

DefineGrid:
    vInput = InputBox("Enter size of grid, as a whole number" & String(2, vbCr) & "(e.g. for a 4 x 4 grid, enter
'4')", sTitle, 4)
    If vInput = "" Then Exit Sub
    If Not IsNumeric(vInput) Then GoTo DefineGrid
    iSide = vInput
    If iSide < 3 Or iNum > 10 Then GoTo DefineGrid
    iSize = iSide ^ 2
    ReDim iGrid(1 To iSize)

Initalize:
    InitializeGrid
    If Not IsSolvable Then GoTo Initalize

GetInput:
    vInput = InputBox(ShowGrid & vbCr & "Enter number to move into empty tile", sTitle)
    If vInput = "" Then
        If MsgBox("Are you sure? This will end the current game.", vbExclamation + vbYesNo, sTitle) = vbYes Then
Exit Sub
    End If
    If Not IsNumeric(vInput) Then
        sMessage = "'" & vInput & "' is not a valid tile"
        GoTo GetInput
    End If
    iNum = vInput
    If iNum < 1 Or iNum > iSize - 1 Then
        sMessage = iNum & " is not a valid tile"
        GoTo GetInput
    End If
    i = FindTile(iNum)
    If Not ValidMove(i) Then GoTo GetInput
    MoveTile (i)
    If TestGrid Then
        MsgBox "SUCCESS! You solved the puzzle in " & lMoves & " moves", vbInformation + vbOKOnly, sTitle
    Else
        GoTo GetInput
    End If
End Sub

Function RandomTile() As Integer
    Randomize
    RandomTile = Int(Rnd * iSize) + 1
End Function
```

```
Function GetX(ByVal i As Integer) As Integer
    GetX = Int((i - 1) / iSide) + 1
End Function

Function GetY(ByVal i As Integer) As Integer
    GetY = (i - 1) Mod iSide + 1
End Function

Function GetI(ByVal x As Integer, y As Integer)
    GetI = (x - 1) * iSide + y
End Function

Function InitializeGrid()
    Dim i As Integer
    Dim x As Integer
    Dim y As Integer

    sMessage = "New " & iSide & " x " & iSide & " game started" & vbCr

    For i = 1 To iSize
        iGrid(i) = 0
    Next i
    For i = 1 To iSize - 1
        Do
            x = RandomTile
            If iGrid(x) = 0 Then iGrid(x) = i
        Loop Until iGrid(x) = i
    Next i
    lMoves = 0
End Function

Function IsSolvable() As Boolean
    Dim i As Integer
    Dim j As Integer
    Dim iCount As Integer
    For i = 1 To iSize - 1
        For j = i + 1 To iSize
            If iGrid(j) < iGrid(i) And iGrid(j) > 0 Then iCount = iCount + 1
        Next j
    Next i
    If iSide Mod 2 Then
        IsSolvable = Not iCount Mod 2
    Else
        IsSolvable = iCount Mod 2 = GetX(FindTile(0)) Mod 2
    End If
End Function

Function TestGrid() As Boolean
    Dim i As Integer

    For i = 1 To iSize - 1
        If Not iGrid(i) = i Then
            TestGrid = False
            Exit Function
        End If
    Next i
    TestGrid = True
End Function

Function FindTile(ByVal iNum As Integer) As Integer
    Dim i As Integer
    For i = 1 To iSize
        If iGrid(i) = iNum Then
            FindTile = i
            Exit Function
        End If
    Next i
End Function

Function ValidMove(ByVal i As Integer) As Boolean
    Dim e As Integer
    Dim xDiff As Integer
    Dim yDiff As Integer

    e = FindTile(0)
    xDiff = GetX(i) - GetX(e)
    yDiff = GetY(i) - GetY(e)
```

```
    If xDiff = 0 Then
        If yDiff = 1 Then
            sMessage = "Tile " & iGrid(i) & " was moved left"
            ValidMove = True
        ElseIf yDiff = -1 Then
            sMessage = "Tile " & iGrid(i) & " was moved right"
            ValidMove = True
        End If
    ElseIf yDiff = 0 Then
        If xDiff = 1 Then
            sMessage = "Tile " & iGrid(i) & " was moved up"
            ValidMove = True
        ElseIf xDiff = -1 Then
            sMessage = "Tile " & iGrid(i) & " was moved down"
            ValidMove = True
        End If
    End If
    If Not ValidMove Then sMessage = "Tile " & iGrid(i) & " may not be moved"
End Function

Function MoveTile(ByVal i As Integer)
    Dim e As Integer
    e = FindTile(0)
    iGrid(e) = iGrid(i)
    iGrid(i) = 0
    lMoves = lMoves + 1
End Function

Function ShowGrid()
    Dim x As Integer
    Dim y As Integer
    Dim i As Integer
    Dim sNum As String
    Dim s As String

    For x = 1 To iSide
        For y = 1 To iSide
            sNum = iGrid(GetI(x, y))
            If sNum = "0" Then sNum = ""
            s = s & sNum & vbTab
        Next y
        s = s & vbCr
    Next x
    If Not sMessage = "" Then
        s = s & vbCr & sMessage & vbCr
    End If
    ShowGrid = s
End Function
```

Sample output:

```
10  4   3   14
9   15  1   6
12  11  7   8
2   5       13

New 4 x 4 game started

Enter number to move into empty tile
```

Visual Basic .NET

```
Public Class Board
    Inherits System.Windows.Forms.Form

    Const XbyX = 4
    Const XSize = 60

    Private Empty As New Panel
    Private Tiles As New List(Of Tile)
    Private Moves As Integer
```

```vb
    Public Sub New()
        Me.Text = XbyX ^ 2 - 1 & " Puzzle Game"
        Me.ClientSize = New Size(XbyX * XSize, XbyX * XSize)
        Me.FormBorderStyle = FormBorderStyle.FixedToolWindow
        Restart()
    End Sub

    Public Sub Restart()
        Dim Start As List(Of Integer) = MakeCompleteable(GetRandamStartOrder())

        Empty.SetBounds(((XbyX ^ 2 - 1) Mod XbyX) * XSize, ((XbyX ^ 2 - 1) \ XbyX) * XSize, XSize, XSize)

        Me.Moves = 0
        Me.Tiles.Clear()
        Me.Controls.Clear()
        For No = 0 To XbyX ^ 2 - 2
            Dim Tile As New Tile
            Tile.Text = Start(No)
            Tile.Board = Me
            Tile.SetBounds((No Mod XbyX) * XSize, (No \ XbyX) * XSize, XSize, XSize)
            Me.Tiles.Add(Tile)
            Me.Controls.Add(Tile)
        Next

    End Sub

    Public Sub IsComplete()
        Me.Moves += 1
        If Empty.Left = ((XbyX ^ 2 - 1) Mod XbyX) * XSize AndAlso Empty.Top = ((XbyX ^ 2 - 1) \ XbyX) * XSize Then
            Me.Tiles.Sort()
            For x = 1 To XbyX ^ 2 - 1
                If Not Tiles(x - 1).Text = x Then
                    Exit Sub
                End If
            Next
            MsgBox($"Completed in {Me.Moves} Moves!", MsgBoxStyle.Information, "Winner")
            Restart()
        End If
    End Sub

    Public Class Tile
        Inherits Button
        Implements IComparable(Of Tile)
        Public Board As Board
        Private Sub Tile_Click(sender As Object, e As EventArgs) Handles Me.Click
            With Board.Empty
                If Me.Left = .Left AndAlso (Me.Top + Me.Height = .Top OrElse .Top + .Height = Me.Top) Then
                    Swap()
                ElseIf Me.Top = .Top AndAlso (Me.Left + Me.Width = .Left OrElse .Left + .Width = Me.Left) Then
                    Swap()
                End If
            End With
        End Sub
        Private Sub Swap()
            Dim p = Board.Empty.Location
            Board.Empty.Location = Me.Location
            Me.Location = p
            Board.IsComplete()
        End Sub
        Public Function CompareTo(other As Tile) As Integer Implements IComparable(Of Tile).CompareTo
            Dim Result = Me.Top.CompareTo(other.Top)
            If Result = 0 Then
                Return Me.Left.CompareTo(other.Left)
            End If
            Return Result
        End Function
    End Class

    Public Function GetRandamStartOrder() As List(Of Integer)
        Dim List As New List(Of Integer)
        Dim Random As New Random()
        Do While List.Count < XbyX ^ 2 - 1
            Dim Value As Integer = Random.Next(1, XbyX ^ 2)
            If Not List.Contains(Value) Then
                List.Add(Value)
            End If
        Loop
        Return List
```

```
    End Function

    Public Function MakeCompleteable(List As List(Of Integer)) As List(Of Integer)
        'ToDo
        Return List
    End Function

End Class
```

Visual Prolog

Output:

15 Puzzle in Visual Prolog - video (https://youtu.be/rWy3AX5HjXM)

```
/* ------------------------------------------------------------------------------------------------

                                            Copyright (c) 2004 - Gal Zsolt (CalmoSoft)

   ------------------------------------------------------------------------------------------------ */

implement playDialog
    open core, vpiDomains, vpiOldDomains, resourceIdentifiers

    facts
        thisWin : vpiDomains::windowHandle := erroneous.

    clauses
        show(Parent):-
            _ = vpi::winCreateDynDialog(Parent, controlList, eventHandler, gui_api::lNull).

/* ------------------------------------------------------------------------------------------------ */

    predicates
        eventHandler : vpiDomains::ehandler.
    clauses
        eventHandler(Win, Event) = Result :-
            Result = generatedEventHandler(Win, Event).

/* ------------------------------------------------------------------------------------------------ */


    predicates
        onDestroy : vpiOldDomains::destroyHandler.
    clauses
        onDestroy() = vpiOldDomains::defaultHandling() :-
            thisWin := erroneous.

/* ------------------------------------------------------------------------------------------------ */

    predicates
        onControlCancel : vpiOldDomains::controlHandler.
    clauses
        onControlCancel(_Ctrl, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull) :-
            file::save("game.dba",gameDB),
            retractall(table(_,_)),
            retractall(gameplay(_,_)),
            retractall(empty_cell(_)),
            retractall(cell(_,_)),
            retractall(cellnr(_,_)),
            retractall(good(_,_,_)),
            vpi::winDestroy(thisWin).

/* ------------------------------------------------------------------------------------------------ */

 predicates
        onUpdate : vpiOldDomains::updateHandler.
    clauses
        onUpdate(Rectangle) = vpiOldDomains::defaultHandling():-
            vpi::winClear( thisWin,Rectangle,0x808040),
            !.

/* ------------------------------------------------------------------------------------------------ */
```

```
    class facts - gameDB
        table:(integer,string).
        gameplay:(integer,integer).
        empty_cell:(integer).

    class facts
        cell:(integer,vpiDomains::windowHandle).
        cellnr:(integer,integer).

/* ------------------------------------------------------------------------------------------------------ */

    class facts
        step_nr:integer:=0.

    predicates
        onCreate : vpiOldDomains::createHandler.
    clauses
        onCreate(_CreationData) = vpiOldDomains::defaultHandling():-
            retractall(table(_,_)),
            retractall(gameplay(_,_)),
            retractall(empty_cell(_)),
            retractall(cell(_,_)),
            retractall(cellnr(_,_)),
            retractall(good(_,_,_)),

            scrollbars_init(),
            cellhandle_init(),
            empty:=16,
            good_cell(0),
            table_save(),
            step_nr:=0,
            fail.
        onCreate(_CreationData) = vpiOldDomains::defaultHandling().

/* ------------------------------------------------------------------------------------------------------ */

    facts
        vsbarr : vpiDomains::windowHandle := erroneous.
        vsbarl : vpiDomains::windowHandle := erroneous.
        hsbart : vpiDomains::windowHandle := erroneous.
        hsbarb : vpiDomains::windowHandle := erroneous.

    predicates
        scrollbars_init:().
    clauses
        scrollbars_init():-
            vsbarr := vpi::winGetCtlHandle(thisWin,idc_sbvr),
            vsbarl := vpi::winGetCtlHandle(thisWin,idc_sbvl),
            hsbart := vpi::winGetCtlHandle(thisWin,idc_sbht),
            hsbarb := vpi::winGetCtlHandle(thisWin,idc_sbhb),


            vpi::winSetScrollRange(vsbarr,sb_Ctl,1,4),
            vpi::winSetScrollProportion(vsbarr,sb_Ctl,1),
            vpi::winSetScrollPos(vsbarr,sb_Ctl,4),

            vpi::winSetScrollRange(vsbarl,sb_Ctl,1,4),
            vpi::winSetScrollProportion(vsbarl,sb_Ctl,1),
            vpi::winSetScrollPos(vsbarl,sb_Ctl,4),

            vpi::winSetScrollRange(hsbart,sb_Ctl,1,4),
            vpi::winSetScrollProportion(hsbart,sb_Ctl,1),
            vpi::winSetScrollPos(hsbart,sb_Ctl,4),

            vpi::winSetScrollRange(hsbarb,sb_Ctl,1,4),
            vpi::winSetScrollProportion(hsbarb,sb_Ctl,1),
            vpi::winSetScrollPos(hsbarb,sb_Ctl,4).

/* ------------------------------------------------------------------------------------------------------ */

    class facts
        cell1 : vpiDomains::windowHandle := erroneous.
        cell2 : vpiDomains::windowHandle := erroneous.
        cell3 : vpiDomains::windowHandle := erroneous.
        cell4 : vpiDomains::windowHandle := erroneous.
        cell5 : vpiDomains::windowHandle := erroneous.
        cell6 : vpiDomains::windowHandle := erroneous.
```

```
        cell7 : vpiDomains::windowHandle := erroneous.
        cell8 : vpiDomains::windowHandle := erroneous.
        cell9 : vpiDomains::windowHandle := erroneous.
        cell10 : vpiDomains::windowHandle := erroneous.
        cell11 : vpiDomains::windowHandle := erroneous.
        cell12 : vpiDomains::windowHandle := erroneous.
        cell13 : vpiDomains::windowHandle := erroneous.
        cell14 : vpiDomains::windowHandle := erroneous.
        cell15 : vpiDomains::windowHandle := erroneous.
        cell16 : vpiDomains::windowHandle := erroneous.

    predicates
        cell_handle:().
    clauses
        cell_handle():-
            cell1:=vpi::winGetCtlHandle(thisWin,idc_cell1),
            cell2:=vpi::winGetCtlHandle(thisWin,idc_cell2),
            cell3:=vpi::winGetCtlHandle(thisWin,idc_cell3),
            cell4:=vpi::winGetCtlHandle(thisWin,idc_cell4),
            cell5:=vpi::winGetCtlHandle(thisWin,idc_cell5),
            cell6:=vpi::winGetCtlHandle(thisWin,idc_cell6),
            cell7:=vpi::winGetCtlHandle(thisWin,idc_cell7),
            cell8:=vpi::winGetCtlHandle(thisWin,idc_cell8),
            cell9:=vpi::winGetCtlHandle(thisWin,idc_cell9),
            cell10:=vpi::winGetCtlHandle(thisWin,idc_cell10),
            cell11:=vpi::winGetCtlHandle(thisWin,idc_cell11),
            cell12:=vpi::winGetCtlHandle(thisWin,idc_cell12),
            cell13:=vpi::winGetCtlHandle(thisWin,idc_cell13),
            cell14:=vpi::winGetCtlHandle(thisWin,idc_cell14),
            cell15:=vpi::winGetCtlHandle(thisWin,idc_cell15),
            cell16:=vpi::winGetCtlHandle(thisWin,idc_cell16).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        cellhandle_init:().
    clauses
        cellhandle_init():-
            retractall(cell(_,_)),
            cell_handle(),

            assert(cell(1,cell1)),
            assert(cell(2,cell2)),
            assert(cell(3,cell3)),
            assert(cell(4,cell4)),
            assert(cell(5,cell5)),
            assert(cell(6,cell6)),
            assert(cell(7,cell7)),
            assert(cell(8,cell8)),
            assert(cell(9,cell9)),
            assert(cell(10,cell10)),
            assert(cell(11,cell11)),
            assert(cell(12,cell12)),
            assert(cell(13,cell13)),
            assert(cell(14,cell14)),
            assert(cell(15,cell15)),
            assert(cell(16,cell16)).

/* ------------------------------------------------------------------------------------------------------ */

    class facts
        good_nr:integer:=0.
        good:(integer,integer,integer) nondeterm.
        empty:integer:=16.

    predicates
        good_cell:(integer) multi.
    clauses
        good_cell(16):-!.
        good_cell(Number):-
            NumberNew = Number+1,
            good_nr:=0,
            good_above(NumberNew),                    % movable cells above empty cell
            good_before(NumberNew),                   % movable cells before empty cell
            good_after(NumberNew),                      % movable cells after empty cell
            good_under(NumberNew),                    % movable cells under empty cell
            assert(cellnr(NumberNew,good_nr)),       % number of movable cells around the empty cell
            good_cell(NumberNew).
```

```
/* ---------------------------------------------------------------------------------------------------- */

    predicates
        good_above:(integer).
    clauses
        good_above(Number):-
            Number > 4,                                          % movable cells above empty cell
            good_nr:= good_nr+1,
            assert(good(Number,good_nr,Number-4)),
            fail.
        good_above(_).

/* ---------------------------------------------------------------------------------------------------- */

    predicates
        good_before:(integer).
    clauses
        good_before(Number):-
            (Number mod 4) <> 1,                               % movable cells before empty cell
            good_nr:= good_nr+1,
            assert(good(Number,good_nr,Number-1)),
            fail.
        good_before(_).

/* ---------------------------------------------------------------------------------------------------- */

    predicates
        good_after:(integer).
    clauses
        good_after(Number):-
            (Number mod 4) > 0,                                  % movable cells after empty cell
            good_nr:= good_nr+1,
            assert(good(Number,good_nr,Number+1)),
            fail.
        good_after(_).

/* ---------------------------------------------------------------------------------------------------- */

    predicates
        good_under:(integer).
    clauses
        good_under(Number):-
            Number < 13,                                         % movable cells under empty cell
            good_nr:= good_nr+1,
            assert(good(Number,good_nr,Number+4)),
            fail.
        good_under(_).

/* ---------------------------------------------------------------------------------------------------- */

    predicates
        cell_click:(integer).

    clauses
        cell_click(NrCell):-
            good(empty,_,NrCell),
            cell(empty,EmptyHandle),
            cell(NrCell,NrCellHandle),
            EmptyNr = vpi::winGetText(NrCellHandle),
            vpi::winSetText(EmptyHandle,EmptyNr),
            vpi::winSetText(NrCellHandle,""),
            empty:=NrCell,
            step_nr := step_nr + 1,
            Bingo = vpi::winGetCtlHandle(thisWin,idc_bingo),
            vpi::winSetText(  Bingo, ""),
            vpi::winSetState(Bingo,[wsf_Invisible]),
            bingo(),

            % VerVal = uncheckedConvert(integer, math::floor((empty-1)/4)+1),
            % HorVal = uncheckedConvert(integer, math::floor((empty-1) mod 4)+1),

            VerVal = math::floor((empty-1)/4)+1,
            HorVal = math::floor((empty-1) mod 4)+1,

            vpi::winSetScrollPos(vsbarr,sb_Ctl,VerVal),
            vpi::winSetScrollPos(vsbarl,sb_Ctl,VerVal),
            vpi::winSetScrollPos(hsbart,sb_Ctl,HorVal),
```

```
            vpi::winSetScrollPos(hsbarb,sb_Ctl,HorVal),

            Step = vpi::winGetCtlHandle(thisWin,idc_step),
            vpi::winSetText(Step,toString(step_nr)),

            assert(gameplay(step_nr,NrCell)),

            fail.
        cell_click(_).

/* ---------------------------------------------------------------------------------------------------- */

    predicates
        cell_click_play:(integer).

    clauses
        cell_click_play(NrCell):-
            good(empty,_,NrCell),
            cell(empty,EmptyHandle),
            cell(NrCell,NrCellHandle),
            EmptyNr = vpi::winGetText(NrCellHandle),
            vpi::winSetText(EmptyHandle,EmptyNr),
            vpi::winSetText(NrCellHandle,""),
            empty:=NrCell,
            Bingo = vpi::winGetCtlHandle(thisWin,idc_bingo),
            vpi::winSetText(Bingo, ""),
            vpi::winSetState(Bingo,[wsf_Invisible]),
            bingo(),

            % VerVal = uncheckedConvert(integer,math::floor((empty-1)/4)+1),
            % HorVal = uncheckedConvert(integer,math::floor((empty-1) mod 4)+1),

            VerVal = math::floor((empty-1)/4)+1,
            HorVal = math::floor((empty-1) mod 4)+1,

            vpi::winSetScrollPos(vsbarr,sb_Ctl,VerVal),
            vpi::winSetScrollPos(vsbarl,sb_Ctl,VerVal),
            vpi::winSetScrollPos(hsbart,sb_Ctl,HorVal),
            vpi::winSetScrollPos(hsbarb,sb_Ctl,HorVal),

            programControl::sleep(1000),

            fail.
        cell_click_play(_).

/* ---------------------------------------------------------------------------------------------------- */

    predicates
        onControlmix : vpiOldDomains::controlHandler.
    clauses
        onControlmix(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            step_nr:=0,
            cell_reset(),
            mix_cells(0),
            Play = vpi::winGetCtlHandle(thisWin,idc_play),
            vpi::winSetState(Play,[wsf_Disabled]),
            Bingo = vpi::winGetCtlHandle(thisWin,idc_bingo),
            vpi::winSetText(Bingo, ""),
            vpi::winSetState(Bingo,[wsf_Invisible]),
            step_nr:=0,
            Step = vpi::winGetCtlHandle(thisWin,idc_step),
            vpi::winSetText(Step,toString(step_nr)),
            table_save(),
            retractall(gameplay(_,_)),
            fail.

        onControlmix(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull).

/* ---------------------------------------------------------------------------------------------------- */

    class facts
        rand_nr:integer:=0.
        mix_nr:integer:=300.

    predicates
        mix_cells:(integer) multi.

    clauses
```

```
        mix_cells(mix_nr):-!.
        mix_cells(Number):-
            NumberNew = Number+1,
            cellnr(empty,CellNr),
            RandomNr = (math::random(1315) mod CellNr) + 1,
            rand_nr := RandomNr,
            good(empty,rand_nr, I),
            cell_click(I),
            mix_cells(NumberNew).
        mix_cells(_).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        table_save:().
    clauses
        table_save():-
            retractall(table(_,_)),
            retractall(empty_cell(_)),
            assert(empty_cell(empty)),

            assert(table(1,vpi::winGetText(cell1))),
            assert(table(2,vpi::winGetText(cell2))),
            assert(table(3,vpi::winGetText(cell3))),
            assert(table(4,vpi::winGetText(cell4))),
            assert(table(5,vpi::winGetText(cell5))),
            assert(table(6,vpi::winGetText(cell6))),
            assert(table(7,vpi::winGetText(cell7))),
            assert(table(8,vpi::winGetText(cell8))),
            assert(table(9,vpi::winGetText(cell9))),
            assert(table(10,vpi::winGetText(cell10))),
            assert(table(11,vpi::winGetText(cell11))),
            assert(table(12,vpi::winGetText(cell12))),
            assert(table(13,vpi::winGetText(cell13))),
            assert(table(14,vpi::winGetText(cell14))),
            assert(table(15,vpi::winGetText(cell15))),
            assert(table(16,vpi::winGetText(cell16))).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlreset : vpiOldDomains::controlHandler.
    clauses
        onControlreset(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
                                cell_reset().

/* ------------------------------------------------------------------------------------------------------ */
    predicates
        cell_reset:().
    clauses
        cell_reset():-
            vpi::winSetText(cell1,"1"),
            vpi::winSetText(cell2,"2"),
            vpi::winSetText(cell3,"3"),
            vpi::winSetText(cell4,"4"),
            vpi::winSetText(cell5,"5"),
            vpi::winSetText(cell6,"6"),
            vpi::winSetText(cell7,"7"),
            vpi::winSetText(cell8,"8"),
            vpi::winSetText(cell9,"9"),
            vpi::winSetText(cell10,"10"),
            vpi::winSetText(cell11,"11"),
            vpi::winSetText(cell12,"12"),
            vpi::winSetText(cell13,"13"),
            vpi::winSetText(cell14,"14"),
            vpi::winSetText(cell15,"15"),
            vpi::winSetText(cell16,""),

            vpi::winSetScrollPos(vsbarr,sb_Ctl,4),
            vpi::winSetScrollPos(vsbarl,sb_Ctl,4),
            vpi::winSetScrollPos(hsbart,sb_Ctl,4),
            vpi::winSetScrollPos(hsbarb,sb_Ctl,4),

            empty:=16,
            step_nr:=0,
            Step = vpi::winGetCtlHandle(thisWin,idc_step),
            vpi::winSetText(Step,toString(step_nr)),
            Bingo = vpi::winGetCtlHandle(thisWin,idc_bingo),
```

```
            vpi::winSetText(Bingo, ""),
            vpi::winSetState(Bingo,[wsf_Invisible]),
            Play = vpi::winGetCtlHandle(thisWin,idc_play),
            vpi::winSetState(Play,[wsf_Disabled]),
            retractall(gameplay(_,_)),
            table_save().

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        bingo:().

    clauses
        bingo():-
            toString(1) = vpi::winGetText(cell1),
            toString(2) = vpi::winGetText(cell2),
            toString(3) = vpi::winGetText(cell3),
            toString(4) = vpi::winGetText(cell4),
            toString(5) = vpi::winGetText(cell5),
            toString(6) = vpi::winGetText(cell6),
            toString(7) = vpi::winGetText(cell7),
            toString(8) = vpi::winGetText(cell8),
            toString(9) = vpi::winGetText(cell9),
            toString(10) = vpi::winGetText(cell10),
            toString(11) = vpi::winGetText(cell11),
            toString(12) = vpi::winGetText(cell12),
            toString(13) = vpi::winGetText(cell13),
            toString(14) = vpi::winGetText(cell14),
            toString(15) = vpi::winGetText(cell15),
            "" = vpi::winGetText(cell16),

            Bingo = vpi::winGetCtlHandle(thisWin,idc_bingo),
            vpi::winSetState(Bingo,[wsf_Visible]),
            vpi::winSetText(Bingo, "BINGO !"),

            Step = vpi::winGetCtlHandle(thisWin,idc_step),
            vpi::winSetText(Step,toString(step_nr)),

            fail.
 bingo().

/* ------------------------------------------------------------------------------------------------------ */

    facts
        fileName:string:="".

    predicates
        onControlsave : vpiOldDomains::controlHandler.
    clauses
        onControlsave(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            try
                FileName = vpiCommonDialogs::getFileName(fileName, ["All files", "*.game"], "Save game as",
[dlgfn_Save], "", _)
            catch _ do
                fail
            end try,
            !,
            file::save(FileName,gameDB).

        onControlsave(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo)= vpiOldDomains::handled(gui_api::rNull).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlopen : vpiOldDomains::controlHandler.
    clauses
        onControlopen(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            try
                FileName = vpiCommonDialogs::getFileName("game_", ["All files", "*.game"], "Open game file", [],
"", _)
            catch _ do
                fail
            end try,
            !,
            retractall(table(_,_)),
            retractall(gameplay(_,_)),
            retractall(empty_cell(_)),
            file::consult(FileName,gameDB),
```

```
            play_display(),
            Play = vpi::winGetCtlHandle(thisWin,idc_play),
            vpi::winSetState(Play,[wsf_Enabled]),
            Bingo = vpi::winGetCtlHandle(thisWin,idc_bingo),
            vpi::winSetState(Bingo,[wsf_Invisible]),
            vpi::winSetText(Bingo, ""),
            step_nr:=0,
            Step = vpi::winGetCtlHandle(thisWin,idc_step),
            vpi::winSetText(Step,toString(step_nr)).

        onControlopen(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull).

/* ------------------------------------------------------------------------------------------------ */

    predicates
        play_display:().
    clauses
        play_display():-

            table(1,Cell1),
            table(2,Cell2),
            table(3,Cell3),
            table(4,Cell4),
            table(5,Cell5),
            table(6,Cell6),
            table(7,Cell7),
            table(8,Cell8),
            table(9,Cell9),
            table(10,Cell10),
            table(11,Cell11),
            table(12,Cell12),
            table(13,Cell13),
            table(14,Cell14),
            table(15,Cell15),
            table(16,Cell16),

            vpi::winSetText(cell1,Cell1),
            vpi::winSetText(cell2,Cell2),
            vpi::winSetText(cell3,Cell3),
            vpi::winSetText(cell4,Cell4),
            vpi::winSetText(cell5,Cell5),
            vpi::winSetText(cell6,Cell6),
            vpi::winSetText(cell7,Cell7),
            vpi::winSetText(cell8,Cell8),
            vpi::winSetText(cell9,Cell9),
            vpi::winSetText(cell10,Cell10),
            vpi::winSetText(cell11,Cell11),
            vpi::winSetText(cell12,Cell12),
            vpi::winSetText(cell13,Cell13),
            vpi::winSetText(cell14,Cell14),
            vpi::winSetText(cell15,Cell15),
            vpi::winSetText(cell16,Cell16),

            empty_cell(Empty_Cell),
            empty:=Empty_Cell,

            fail.
        play_display().

/* ------------------------------------------------------------------------------------------------ */

    predicates
        onControlplay : vpiOldDomains::controlHandler.
    clauses

        onControlplay(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            Play = vpi::winGetCtlHandle(thisWin,idc_play),
            vpi::winSetState(Play,[wsf_Disabled]),
            gameplay(Nr,Nr_cell),
            cell_click_play(Nr_cell),
            Step = vpi::winGetCtlHandle(thisWin,idc_step),
            vpi::winSetText(Step,toString(Nr)),
            fail.

        onControlplay(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull).

/* ------------------------------------------------------------------------------------------------ */
```

```
    predicates
        onControlsbvr : vpiOldDomains::controlHandler.              % Sets the right vertical scrollbar
    clauses
        onControlsbvr(_CtrlID, _CtrlType, _CtrlWin,scroll(sc_LineDown,_)) =
vpiOldDomains::handled(gui_api::rNull):-
            Sbarvalr = vpi::winGetScrollPos(vsbarr,sb_Ctl)+1,
            Sbarvalr < 5,
            cell_click(empty+4),
            vpi::winSetScrollPos(vsbarr,sb_Ctl, Sbarvalr),
            vpi::winSetScrollPos(vsbarl,sb_Ctl, Sbarvalr),
            fail.

        onControlsbvr(_CtrlID, _CtrlType, _CtrlWin,scroll(sc_LineUp,_)) = vpiOldDomains::handled(gui_api::rNull):-
            Sbarvalr = vpi::winGetScrollPos(vsbarr,sb_Ctl)-1,
            Sbarvalr > 0,
            cell_click(empty-4),
            vpi::winSetScrollPos(vsbarr,sb_Ctl, Sbarvalr),
            vpi::winSetScrollPos(vsbarl,sb_Ctl, Sbarvalr),
            fail.

        onControlsbvr(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull).


/* ------------------------------------------------------------------------------------------------ */

    predicates
        onControlsbvl : vpiOldDomains::controlHandler.                  % Sets the left vertical scrollbar
    clauses
        onControlsbvl(_CtrlID, _CtrlType, _CtrlWin,scroll(sc_LineDown,_)) =
vpiOldDomains::handled(gui_api::rNull):-
            Sbarvall = vpi::winGetScrollPos(vsbarl,sb_Ctl)+1,
            Sbarvall < 5,
            cell_click(empty+4),
            vpi::winSetScrollPos(vsbarl,sb_Ctl, Sbarvall),
            vpi::winSetScrollPos(vsbarr,sb_Ctl, Sbarvall),
            fail.

        onControlsbvl(_CtrlID, _CtrlType, _CtrlWin,scroll(sc_LineUp,_)) = vpiOldDomains::handled(gui_api::rNull):-
            Sbarvall = vpi::winGetScrollPos(vsbarl,sb_Ctl)-1,
            Sbarvall > 0,
            cell_click(empty-4),
            vpi::winSetScrollPos(vsbarl,sb_Ctl, Sbarvall),
            vpi::winSetScrollPos(vsbarr,sb_Ctl, Sbarvall),
            fail.

        onControlsbvl(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull).

/* ------------------------------------------------------------------------------------------------ */

    predicates
        onControlsbht : vpiOldDomains::controlHandler.              % Sets the top horizontal scrollbar
    clauses
        onControlsbht(_CtrlID, _CtrlType, _CtrlWin,scroll(sc_LineDown,_)) =
vpiOldDomains::handled(gui_api::rNull):-
            Sbarvalt = vpi::winGetScrollPos(hsbart,sb_Ctl)+1,
            Sbarvalt < 5,
            cell_click(empty+1),
            vpi::winSetScrollPos(hsbart,sb_Ctl, Sbarvalt),
            vpi::winSetScrollPos(hsbarb,sb_Ctl, Sbarvalt),
            fail.

        onControlsbht(_CtrlID, _CtrlType, _CtrlWin,scroll(sc_LineUp,_)) = vpiOldDomains::handled(gui_api::rNull):-
            Sbarvalt = vpi::winGetScrollPos(hsbart,sb_Ctl)-1,
            Sbarvalt > 0,
            cell_click(empty-1),
            vpi::winSetScrollPos(hsbart,sb_Ctl, Sbarvalt),
            vpi::winSetScrollPos(hsbarb,sb_Ctl, Sbarvalt),
            fail.

        onControlsbht(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull).

/* ------------------------------------------------------------------------------------------------ */

    predicates
        onControlsbhb : vpiOldDomains::controlHandler.       % Sets the bottom horizontal scrollbar
    clauses
        onControlsbhb(_CtrlID, _CtrlType, _CtrlWin,scroll(sc_LineDown,_)) =
vpiOldDomains::handled(gui_api::rNull):-
```

```
            Sbarvalb = vpi::winGetScrollPos(hsbarb,sb_Ctl)+1,
            Sbarvalb < 5,
            cell_click(empty+1),
            vpi::winSetScrollPos(hsbarb,sb_Ctl, Sbarvalb),
            vpi::winSetScrollPos(hsbart,sb_Ctl, Sbarvalb),
            fail.

        onControlsbhb(_CtrlID, _CtrlType, _CtrlWin,scroll(sc_LineUp,_)) = vpiOldDomains::handled(gui_api::rNull):-
            Sbarvalb = vpi::winGetScrollPos(hsbarb,sb_Ctl)-1,
            Sbarvalb > 0,
            cell_click(empty-1),
            vpi::winSetScrollPos(hsbarb,sb_Ctl, Sbarvalb),
            vpi::winSetScrollPos(hsbart,sb_Ctl, Sbarvalb),
            fail.

        onControlsbhb(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell1 : vpiOldDomains::controlHandler.
    clauses
        onControlcell1(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(1).

/* ------------------------------------------------------------------------------------------------------ */


    predicates
        onControlcell2 : vpiOldDomains::controlHandler.
    clauses
        onControlcell2(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(2).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell3 : vpiOldDomains::controlHandler.
    clauses
        onControlcell3(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(3).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell4 : vpiOldDomains::controlHandler.
    clauses
        onControlcell4(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(4).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell5 : vpiOldDomains::controlHandler.
    clauses
        onControlcell5(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(5).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell6 : vpiOldDomains::controlHandler.
    clauses
        onControlcell6(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(6).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell7 : vpiOldDomains::controlHandler.
    clauses
        onControlcell7(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(7).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell8 : vpiOldDomains::controlHandler.
```

```
    clauses
        onControlcell8(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(8).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell9 : vpiOldDomains::controlHandler.
    clauses
        onControlcell9(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(9).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell10 : vpiOldDomains::controlHandler.
    clauses
        onControlcell10(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(10).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell11 : vpiOldDomains::controlHandler.
    clauses
        onControlcell11(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(11).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell12 : vpiOldDomains::controlHandler.
    clauses
        onControlcell12(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(12).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell13 : vpiOldDomains::controlHandler.
    clauses
        onControlcell13(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(13).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell14 : vpiOldDomains::controlHandler.
    clauses
        onControlcell14(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(14).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell15 : vpiOldDomains::controlHandler.
    clauses
        onControlcell15(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(15).

/* ------------------------------------------------------------------------------------------------------ */

    predicates
        onControlcell16 : vpiOldDomains::controlHandler.
    clauses
        onControlcell16(_CtrlID, _CtrlType, _CtrlWin, _CtrlInfo) = vpiOldDomains::handled(gui_api::rNull):-
            cell_click(16).

/* ------------------------------------------------------------------------------------------------------ */

    % This code is maintained by the VDE do not update it manually, 22:12:35-14.3.2004
    constants
        dialogType : vpiDomains::wintype = wd_Modal.
        title : string = "playDialog".
        rectangle : vpiDomains::rct = rct(200,40,359,263).
        dialogFlags : vpiDomains::wsflags = [wsf_Close,wsf_TitleBar].
        dialogFont = "MS Sans Serif".
        dialogFontSize = 8.
```

```
    constants
        controlList : vpiDomains::windef_list =
            [
            dlgFont(wdef(dialogType, rectangle, title, u_DlgBase),
                    dialogFont, dialogFontSize, dialogFlags),
            ctl(wdef(wc_PushButton,rct(40,166,67,178),"&Mix",u_DlgBase),idc_mix,[wsf_Group,wsf_TabStop]),
            ctl(wdef(wc_PushButton,rct(67,166,94,178),"&Reset",u_DlgBase),idc_reset,[wsf_Group,wsf_TabStop]),
            ctl(wdef(wc_PushButton,rct(40,178,67,190),"&Save",u_DlgBase),idc_save,[wsf_Group,wsf_TabStop]),
            ctl(wdef(wc_PushButton,rct(94,178,121,190),"&Open",u_DlgBase),idc_open,[wsf_Group,wsf_TabStop]),
            ctl(wdef(wc_PushButton,rct(40,190,121,202),"&Play",u_DlgBase),idc_play,
[wsf_Group,wsf_TabStop,wsf_Disabled]),
            ctl(wdef(wc_PushButton,rct(94,166,121,178),"&Exit",u_DlgBase),idc_cancel,[wsf_Group,wsf_TabStop]),
            ctl(wdef(wc_PushButton,rct(40,40,60,60),"1",u_DlgBase),idc_cell1,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(60,40,80,60),"2",u_DlgBase),idc_cell2,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(80,40,100,60),"3",u_DlgBase),idc_cell3,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(100,40,120,60),"4",u_DlgBase),idc_cell4,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(40,60,60,80),"5",u_DlgBase),idc_cell5,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(60,60,80,80),"6",u_DlgBase),idc_cell6,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(80,60,100,80),"7",u_DlgBase),idc_cell7,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(100,60,120,80),"8",u_DlgBase),idc_cell8,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(40,80,60,100),"9",u_DlgBase),idc_cell9,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(60,80,80,100),"10",u_DlgBase),idc_cell10,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(80,80,100,100),"11",u_DlgBase),idc_cell11,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(100,80,120,100),"12",u_DlgBase),idc_cell12,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(40,100,60,120),"13",u_DlgBase),idc_cell13,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(60,100,80,120),"14",u_DlgBase),idc_cell14,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(80,100,100,120),"15",u_DlgBase),idc_cell15,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(100,100,120,120),"",u_DlgBase),idc_cell16,[wsf_Group]),
            ctl(wdef(wc_HScroll,rct(30,18,130,30),"",u_DlgBase),idc_sbht,[]),
            ctl(wdef(wc_VScroll,rct(130,30,142,130),"",u_DlgBase),idc_sbvr,[]),
            ctl(wdef(wc_HScroll,rct(30,130,130,142),"",u_DlgBase),idc_sbhb,[]),
            ctl(wdef(wc_VScroll,rct(18,30,30,130),"",u_DlgBase),idc_sbvl,[]),
            ctl(wdef(wc_PushButton,rct(67,178,94,190),"",u_DlgBase),idc_step,[wsf_Group]),
            ctl(wdef(wc_PushButton,rct(40,154,121,166),"",u_DlgBase),idc_bingo,[wsf_Group,wsf_Invisible])
            ].

    predicates
        generatedEventHandler : vpiDomains::ehandler.
    clauses
        generatedEventHandler(Win, e_create(_)) = _ :-
            thisWin := Win,
            fail.
        generatedEventHandler(_Win, e_Create(CreationData)) = Result :-
            handled(Result) = onCreate(CreationData).
        generatedEventHandler(_Win, e_Update(Rectangle)) = Result :-
            handled(Result) = onUpdate(Rectangle).
        generatedEventHandler(_Win, e_Control(idc_cancel, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlCancel(idc_cancel, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_mix, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlmix(idc_mix, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_reset, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlreset(idc_reset, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_sbvr, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlsbvr(idc_sbvr, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_sbvl, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlsbvl(idc_sbvl, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_sbhb, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlsbhb(idc_sbhb, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_sbht, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlsbht(idc_sbht, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_save, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlsave(idc_save, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_open, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlopen(idc_open, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_play, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlplay(idc_play, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell1, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell1(idc_cell1, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell10, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell10(idc_cell10, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell11, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell11(idc_cell11, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell12, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell12(idc_cell12, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell13, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell13(idc_cell13, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell14, CtrlType, CtrlWin, CtlInfo)) = Result :-
```

```
            handled(Result) = onControlcell14(idc_cell14, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell15, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell15(idc_cell15, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell16, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell16(idc_cell16, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell2, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell2(idc_cell2, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell3, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell3(idc_cell3, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell4, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell4(idc_cell4, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell5, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell5(idc_cell5, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell6, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell6(idc_cell6, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell7, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell7(idc_cell7, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell8, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell8(idc_cell8, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Control(idc_cell9, CtrlType, CtrlWin, CtlInfo)) = Result :-
            handled(Result) = onControlcell9(idc_cell9, CtrlType, CtrlWin, CtlInfo).
        generatedEventHandler(_Win, e_Destroy()) = Result :-
            handled(Result) = onDestroy().
    % end of automatic code
end implement playDialog
```

VBScript

```
'----------------15 game--------------------------------------
'WARNING: this script uses ANSI escape codes to position items on console so
'it won't work in Windows from XP to 8.1 where Microsoft removed ANSI support...
'Windows 10, 11 or 98 are ok!!!

option explicit
const maxshuffle=100 'level

dim ans0:ans0=chr(27)&"["
dim anscls:anscls=ans0 & "2J"

dim dirs:dirs=array(6,-1,-6,1)
dim a:a=array(-1,-1,-1,-1,-1,-1,_
              -1, 1, 2, 3, 4,-1,_
              -1, 5, 6, 7, 8,-1,_
              -1, 9,10,11,12,-1,_
              -1,13,14,15, 0,-1,_
              -1,-1,-1,-1,-1,-1)
dim b(35)
dim pos0
dim s:s=Array("W+Enter: up   Z+Enter: down  A+Enter: left   S+Enter right ",_
              "Bad move!!                                                   ",_
          "You did it! Another game? [y/n]+Enter                        ",_
          "Bye!                                                         ")


do
  shuffle
  draw
  toxy 10,1,s(0)
  do
   if usr(wait()) then
     draw
     toxy 10,1,s(0)
   else
     toxy 10,1,s(1)
   end if
  loop until checkend
  toxy 10,1,s(2)
loop until wait()="n"
toxy 10,1,s(3)

function wait():
 toxy 11,1,"":
 wait=left(lcase(wscript.stdin.readline),1):
end function
```

```
sub toxy(x,y,s)
 wscript.StdOut.Write ans0 & x & ";" & y & "H"& " "& s
end sub

sub draw
  dim i,j
  wscript.stdout.write anscls
  for i=0 to 3 'row
    for j=0 to 3  'col
      toxy (j*2)+2,(i*3)+3,iif(b(j*6+i+7)<>0,b(j*6+i+7)," ")
    next
  next
  toxy 10,1,""
end sub


function checkend
  dim i
  for i=0 to ubound(a)
   if (b(i)<>a(i)) then checkend=false : exit function
  next
  checkend=true
end function

function move(d)
 dim p1
 p1=pos0+d
 if b(p1) <>-1 then
    b(pos0)=b(p1):
    b(p1)=0:
    pos0=p1:
    move=true
 else
   move=false
 end if
end function

sub shuffle
  dim cnt,r,i
   randomize timer
   for i=0 to ubound(a):b(i)=a(i):next
   pos0=28
   cnt=0
   do
     r=int(rnd*4)
     if move(dirs(r))=true then cnt=cnt+1
   loop until cnt=maxshuffle
end sub

function iif(a,b,c): if a then iif=b else iif=c end if:end function

function usr(a)
  dim d
  select case lcase(a)
  case "w" :d=-6
  case "z" :d=6
  case "a" :d=-1
  case "s" :d=1
  end select
  usr= move(d)
end function
```

Wren

Translation of: Go
Library: Wren-dynamic
Library: Wren-ioutil
Library: Wren-fmt

```
import "random" for Random
import "/dynamic" for Enum
import "/ioutil" for Input
```

```
import "/fmt" for Fmt

var Move = Enum.create("Move", ["up", "down", "right", "left"])

var Rand = Random.new()
var RandMove = Fn.new { Rand.int(4) }

var SolvedBoard = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]

var AreEqual = Fn.new { |l1, l2|
    if (l1.count != l2.count) return false
    for (i in 0...l1.count) {
        if (l1[i] != l2[i]) return false
    }
    return true
}

class Puzzle {
    construct new() {
        _board = SolvedBoard.toList
        _empty = 15 // _board[empty] == 0
        _moves = 0
        _quit  = false
        // Could make this configurable, 10 == easy, 50 == normal, 100 == hard
        shuffle(50)
    }

    shuffle(moves) {
        // we use some number of random moves to "shuffle" the board
        var i = 0
        while (i < moves || AreEqual.call(_board, SolvedBoard)) {
            if (doMove(RandMove.call())) i = i + 1
        }
    }

    isValidMove(m) {
        if (m == Move.up)    return [_empty - 4, (_empty/4).floor > 0]
        if (m == Move.down)  return [_empty + 4, (_empty/4).floor < 3]
        if (m == Move.right) return [_empty + 1, _empty % 4 < 3]
        if (m == Move.left)  return [_empty - 1, _empty % 4 > 0]
        Fiber.abort("not reached")
    }

    doMove(m) {
        var i = _empty
        var res = isValidMove(m)
        var j  = res[0]
        var ok = res[1]
        if (ok) {
            _board.swap(i, j)
            _empty = j
            _moves = _moves + 1
        }
        return ok
    }

    play() {
        var instructions = """
Please enter "U", "D", "L", or "R" to move the empty cell
up, down, left, or right. You can also enter "Q" to quit.
Upper or lowercase is accepted and only the first character
is important (i.e. you may enter "up" if you like).
"""
        System.print(instructions)
        System.write("\nStarting board:")
        while (!AreEqual.call(_board, SolvedBoard) && !_quit) {
            System.print("\n%(this)")
            playOneMove()
        }
        if (AreEqual.call(_board, SolvedBoard)) {
            System.print("\n%(this)")
            System.print("You solved the puzzle in %(_moves) moves.")
        }
    }

    playOneMove() {
        while (true) {
            var s = Input.option("Enter move #%(_moves + 1) (U, D, L, R or Q): ", "UuDdLlRrQq")
```

```
            var m
            if (s == "U" || s == "u") {
                m = Move.up
            } else if (s == "D" || s == "d") {
                m = Move.down
            } else if (s == "L" || s == "l") {
                m = Move.left
            } else if (s == "R" || s == "r") {
                m = Move.right
            } else if (s == "Q" || s == "q") {
                System.print("Quiting after %(_moves).")
                _quit = true
                return
            }
            if (!doMove(m)) {
                System.print("That is not a valid move at the moment.")
                continue
            }
            return
        }
    }

    toString {
        var buf = ""
        var i = 0
        for (c in _board) {
            if (c == 0) {
                buf = buf + "  ."
            } else {
                buf = buf + Fmt.swrite("$3d", c)
            }
            if (i % 4 == 3) buf = buf + "\n"
            i = i + 1
        }
        return buf
    }
}

var p = Puzzle.new()
p.play()
```

Output:

Sample (very easy!) game:

```
Please enter "U", "D", "L", or "R" to move the empty cell
up, down, left, or right. You can also enter "Q" to quit.
Upper or lowercase is accepted and only the first character
is important (i.e. you may enter "up" if you like).

Starting board:
  1  2  3  4
  6  .  7  8
  5  9 11 12
 13 10 14 15

Enter move #51 (U, D, L, R or Q): l

  1  2  3  4
  .  6  7  8
  5  9 11 12
 13 10 14 15

Enter move #52 (U, D, L, R or Q): d

  1  2  3  4
  5  6  7  8
  .  9 11 12
 13 10 14 15

Enter move #53 (U, D, L, R or Q): r

  1  2  3  4
  5  6  7  8
  9  . 11 12
```

```
 13 10 14 15

Enter move #54 (U, D, L, R or Q): d

  1  2  3  4
  5  6  7  8
  9 10 11 12
 13  . 14 15

Enter move #55 (U, D, L, R or Q): r

  1  2  3  4
  5  6  7  8
  9 10 11 12
 13 14  . 15

Enter move #56 (U, D, L, R or Q): r

  1  2  3  4
  5  6  7  8
  9 10 11 12
 13 14 15  .

You solved the puzzle in 56 moves.
```

x86-64 Assembly

```
        ; Puzzle15 by grosged (march 2019)
        ; How to play ?.. Just press one of the arrow keys then [enter] to valid
        ; ( press [Ctrl+C] to escape )

        segment .data
check:  db  "1   2   3   4",10,"  5   6   7   8",10,"  9  10  11  12",10," 13  14  1"
puzzle: db  27,"c",10,"  1   2   3   4",10,"  5   6   7   8",10,"  9  10  11  12",10," 13  14  15    ",10,10
    db  " Direction ?",13
    db  " Well done !  ",10,10
inKey:  dw  0,0,0,0

    segment .text
    global  _start

_start: mov rax,100
    syscall
    mov rcx,rax
    shr rcx,3
    and rcx,255
    and rax,31
    lea rsi,[_start+rax]
    mov rbx,15

Mixing: push    rcx
    mov di,word[rsi+rcx*2]
    mov rcx,8
quatre: mov ax,di
    shr di,2
    and ax,3
    add ax,65
    call    SWAPPIN
    loop    quatre
    pop rcx
    loop    Mixing
    cmp cx,ax

MainLp: mov rdx,80
Succes: lea rsi,[puzzle]
    mov rdi,1
    mov rax,1
End?:   syscall
    mov rax,60
    je  End?
    mov rdx,8
    lea rsi,[inKey]
    mov rdi,0
    mov rax,0
```

```
    syscall
    mov al,byte [rsi+rax-2]
    call    SWAPPIN
    lea     rsi,[check]
    lea rdi,[puzzle+5]
    mov rcx,7
    repe    cmpsq
    jne MainLp
    mov rdx,95
    jmp Succes

SWAPPIN:mov rdx,rbx
    cmp al,"A"
    jne NotUp
    add rbx,4
    cmp rbx,16
    cmovae  rbx,rdx
NotUp:  cmp al,"B"
    jne NotDwn
    sub rbx,4
    cmovc   rbx,rdx
NotDwn: cmp al,"C"
    jne NotLft
    test    rbx,3
    jz  Endcll
    dec rbx
NotLft: cmp al,"D"
    jne Endcll
    inc rbx
    test    rbx,3
    cmovz   rbx,rdx
Endcll: mov ax,"  "
    xchg    ax,word[puzzle+4+rbx*4]
    mov word[puzzle+4+rdx*4],ax
    ret
```

XBasic

Translation of: Liberty BASIC
Works with: Windows XBasic

```
PROGRAM "fifteenpuzzlegame"
VERSION "0.0001"

IMPORT "xst"

DECLARE FUNCTION Entry()
INTERNAL FUNCTION PrintPuzzle(d%[])
INTERNAL FUNCTION IntroAndLevel(shCnt%[])
INTERNAL FUNCTION BuildBoard(d%[], shCnt%[], level%)
INTERNAL FUNCTION IsMoveValid(d%[], piece%, piecePos%, emptyPos%)
INTERNAL FUNCTION IsPuzzleComplete(d%[])
INTERNAL FUNCTION PiecePosition(d%[], piece%)

' Pseudo-random number generator
' Based on the rand, srand functions from Kernighan & Ritchie's book
' 'The C Programming Language'
DECLARE FUNCTION Rand()
DECLARE FUNCTION SRand(seed%%)

FUNCTION Entry()
  DIM d%[15]
  DIM shCnt%[2]
  BuildBoard(@d%[], @shCnt%[], IntroAndLevel(@shCnt%[]))
  PrintPuzzle(@d%[])
  DO
    x% = SSHORT(INLINE$("To move a piece, enter its number: "))
    DO WHILE IsMoveValid(@d%[], x%, @y%, @z%) = 0
      PRINT "Wrong move."
      PrintPuzzle(@d%[])
      x% = SSHORT(INLINE$("To move a piece, enter its number: "))
    LOOP
    d%[z% - 1] = x%
    d%[y% - 1] = 0
```

```
    PrintPuzzle(@d%[])
  LOOP UNTIL IsPuzzleComplete(@d%[])
  PRINT "YOU WON!"
END FUNCTION

FUNCTION PrintPuzzle(d%[])
  DIM ds$[15] ' Board pieces
  FOR p%% = 0 TO 15
    IF d%[p%%] = 0 THEN
      ds$[p%%] = "     "
    ELSE
      ds$[p%%] = FORMAT$("###  ", d%[p%%])
    END IF
  NEXT p%%
  PRINT "+-----+-----+-----+-----+"
  PRINT "|"; ds$[0]; "|"; ds$[1]; "|"; ds$[2]; "|"; ds$[3]; "|"
  PRINT "+-----+-----+-----+-----+"
  PRINT "|"; ds$[4]; "|"; ds$[5]; "|"; ds$[6]; "|"; ds$[7]; "|"
  PRINT "+-----+-----+-----+-----+"
  PRINT "|"; ds$[8]; "|"; ds$[9]; "|"; ds$[10]; "|"; ds$[11]; "|"
  PRINT "+-----+-----+-----+-----+"
  PRINT "|"; ds$[12]; "|"; ds$[13]; "|"; ds$[14]; "|"; ds$[15]; "|"
  PRINT "+-----+-----+-----+-----+"
END FUNCTION

FUNCTION IntroAndLevel(shCnt%[])
  XstClearConsole()
  shCnt%[0] = 10
  shCnt%[1] = 50
  shCnt%[2] = 100
  PRINT "15 PUZZLE GAME"
  PRINT
  PRINT
  PRINT "Please enter level of difficulty,"
  DO
    level% = SSHORT(INLINE$("1 (easy), 2 (medium) or 3 (hard): "))
  LOOP UNTIL (level% >= 1) && (level% <= 3)
END FUNCTION level%

FUNCTION BuildBoard(d%[], shCnt%[], level%)
  ' Set pieces in correct order first
  FOR p%% = 1 TO 15
    d%[p%% - 1] = p%%
  NEXT p%%
  d%[15] = 0 ' 0 = empty piece/slot
  z% = 16 ' z% = empty position
  PRINT
  PRINT "Shuffling pieces";
  XstGetSystemTime (@msec)
  SRand(INT(msec) MOD 32768)
  FOR n% = 1 TO shCnt%[level% - 1]
    PRINT ".";
    DO
      x% = INT(Rand() / 32768.0 * 4.0) + 1
      PRINT x%
      SELECT CASE x%
        CASE 1:
          r% = z% - 4
        CASE 2:
          r% = z% + 4
        CASE 3:
          IF (z% - 1) MOD 4 <> 0 THEN
            r% = z% - 1
          END IF
        CASE 4:
          IF z% MOD 4 <> 0 THEN
            r% = z% + 1
          END IF
      END SELECT
    LOOP UNTIL (r% >= 1) && (r% <= 16)
    d%[z% - 1] = d%[r% - 1]
    z% = r%
    d%[z% - 1] = 0
  NEXT n%
  XstClearConsole()
END FUNCTION

FUNCTION IsMoveValid(d%[], piece%, piecePos%, emptyPos%)
```

```
  mv% = 0
  IF (piece% >= 1) && (piece% <= 15) THEN
    piecePos% = PiecePosition(@d%[], piece%)
    emptyPos% = PiecePosition(@d%[], 0)
    ' Check if empty piece is above, below, left or right to piece
    IF (piecePos% - 4 = emptyPos%) || (piecePos% + 4 = emptyPos%) || ((piecePos% - 1 = emptyPos%) && (emptyPos%
MOD 4 <> 0)) || ((piecePos% + 1 = emptyPos%) && (piecePos% MOD 4 <> 0)) THEN
      mv% = 1
    END IF
  END IF
END FUNCTION mv%

FUNCTION PiecePosition(d%[], piece%)
  p%% = 0
  DO WHILE d%[p%%] <> piece%
    INC p%%
    IF p%% > 15 THEN
      PRINT "UH OH!"
      QUIT(1)
    END IF
  LOOP
END FUNCTION p%% + 1

FUNCTION IsPuzzleComplete(d%[])
  pc% = 0
  p%% = 1
  DO WHILE (p%% < 16) && (d%[p%% - 1] = p%%)
    INC p%%
  LOOP
  IF p%% = 16 THEN
    pc% = 1
  END IF
END FUNCTION pc%

' Return pseudo-random integer on 0..32767
FUNCTION Rand()
  #next&& = #next&& * 1103515245 + 12345
END FUNCTION USHORT(#next&& / 65536) MOD 32768

' Set seed for Rand()
FUNCTION SRand(seed%%)
  #next&& = seed%%
END FUNCTION

END PROGRAM
```

XPLo

```
int Box, Hole, I;
[Box:= [^ ,^F,^E,^D,    \starting configuration
        ^C,^B,^A,^9,    \slide digits into ascending order
        ^8,^7,^6,^5,    \with blank in lower-right corner
        ^4,^3,^2,^1];
Hole:= 0;               \index for hole position
loop    [Clear;         \erase screen and move to start
        for I:= 0 to 15 do \show puzzle
                [ChOut(0, Box(I));  ChOut(0, ^ );
                if (I & 3) = 3 then CrLf(0)];
        case ChIn(1) of                 \get move
          $1B:  quit;                   \Esc
          $48:  if Hole < 12 then       \UpArrow scan code
                        [Box(Hole):= Box(Hole+4);
                        Hole:= Hole+4;
                        Box(Hole):= ^ ];
          $50:  if Hole > 3 then        \DnArrow
                        [Box(Hole):= Box(Hole-4);
                        Hole:= Hole-4;
                        Box(Hole):= ^ ];
          $4B:  if (Hole & 3) < 3 then  \LfArrow
                        [Box(Hole):= Box(Hole+1);
                        Hole:= Hole+1;
                        Box(Hole):= ^ ];
          $4D:  if (Hole & 3) > 0 then  \RtArrow
                        [Box(Hole):= Box(Hole-1);
```

```
                        Hole:= Hole-1;
                        Box(Hole):= ^ ]
        other   [];     \ignore 0 scan code prefix etc.
        ];
]
```

Output:

```
  F E D
C B A 9
8 7 6 5
4 3 2 1
```

Yabasic

```
dx = 4 : dy = 4 : dxy = dx * dy
dim grid(dx, dy)

dim t(dxy)

for x = 1 to dx
    for y = 1 to dy
        fin = false
        repeat
            i = int(ran(dxy) + 1)
            if t(i) = 0 then
                t(i) = 1
                fin = true
                if i < dxy then
                    grid(x, y) = i
                else
                    ex = x : ey = y
                end if
            end if
        until(fin = true)
    next y
next x

drawTable()

repeat
    moveTile()
    drawTable()
until(finish())

print "\n\n\nNumber of movements: ",mov,"\nEnd\n"

sub finish()
    local x, y, v

    for y = 1 to dy
        for x = 1 to dx
            v = v + 1
            if (v < dxy) and (grid(x, y) <> v) return false
        next x
    next y
    return true
end sub

sub moveTile()
    local direction$

    direction$ = inkey$

    switch direction$
        case "up": if (ey + 1) < (dy + 1) then grid(ex, ey) = grid(ex, ey + 1) : ey = ey + 1  end if : break
        case "down": if (ey - 1) > 0 then grid(ex, ey) = grid(ex, ey - 1) : ey = ey - 1  end if : break
        case "right": if (ex - 1) > 0 then grid(ex, ey) = grid(ex - 1, ey) : ex = ex - 1  end if : break
        case "left": if (ex + 1) < (dx + 1) then grid(ex, ey) = grid(ex + 1, ey) : ex = ex + 1  end if : break
        default: return : break
    end switch
    mov = mov + 1
```

```
    grid(ex, ey) = 0
end sub

sub drawTable()
    local x, y

    clear screen

    print "   Use the cursor keys"

    for x = 1 to dx
        for y = 1 to dy
            print at(x * 3, y * 2);
            if grid(x, y) then
                print color("yellow","magenta") grid(x, y) using "##"
            else
                print "  "
            end if
        next y
    next x
end sub
```

Adaptation from Phix solution

```
board$ = "123456789ABCDEF0"
solve$ = board$
pos = 16

sub print_board()
    local i, n$

    clear screen
    for i = 1 to len(board$)
        if i = pos then
            print "   ";
        else
            n$ = str$(dec(mid$(board$, i, 1)), "###")
            print n$;
        end if
        if mod(i, 4) = 0 print
    next
    print
end sub

sub move(d)
    local new_pos, delta(4)

    delta(1) = 4 : delta(2) = 1 : delta(3) = -1 : delta(4) = -4

    new_pos = pos + delta(d)
    if new_pos >= 1 and new_pos <= 16 and (mod(pos, 4) = mod(new_pos, 4) or floor((pos - 1) / 4) = floor((new_pos
- 1) / 4)) then
        mid$(board$, pos, 1) = mid$(board$, new_pos, 1)
        mid$(board$, new_pos, 1) = "0"
        pos = new_pos
    end if
end sub

for i = 1 to 100 : move(int(ran(4))+1) : next
do
    print_board()
    if board$ = solve$ break
    c = ((instr("esc  up   left rightdown ", inkey$) - 1) / 5)
    if c < 1 break
    move(c)
loop
print "solved!\n"
```
