```
; ***  P I T F A L L !  ***
; Copyright 1982 Activision
; Designer: David Crane

; Analyzed, labeled and commented
;  by Thomas Jentzsch (JTZ)
; Last Update: 09.10.2001 (v0.9)

; Scene generation:
; The 255 scenes are randomly generated through a "bidirecional" LFSR. This
; means, that the process of generating the next random number is reversible,
; which is necessary to allow traveling though the jungle in both directions.
; The random number defines the scene with the following bits:
; - 0..2: type of object(s) on the ground (logs, fire, cobra, treasure etc.)
; - 3..5: type of scene (holes, pits, crocdiles, treasures etc.)
; - 6..7: type of the tree pattern
; - 7   : position of the wall (left or right)
;
; Kernel:
; The kernel is very large. There are special loops for several horizontal
; stripes. Before the main kernel the game draws the score and the lives with
; the timer. Then follow nine stripes, which do:
; 1. draw the trees and the branches
; 2. draw the top of the liana and x-position Harry and the ground object
; 3. draw Harry and the middle of the liana
; 4. draw Harry and the bottom of the liana
; 5. draw Harry and the top of the ground object
; 6. draw the bottom of the ground object and the top of the holes/pits (no Harry)
; 7. draw Harry, holes/pits and the top of the ladder
; 8. draw Harry, the ladder and the top of the wall
; 9. draw Harry, the bottom of the ladder and the wall or the scorpion
; Finally after that, the copyright message is drawn.
;
; Misc:
; - The game speeds aren't adjusted for PAL, the PAL game run's about 16% slower.
;   (This seems to be true for most PAL conversions.).
; - The timer is also running slower, it would have been quite easy to adjust
;   that (s. FRAMERATE etc.).


    processor 6502
    include vcs.h


;===============================================================================
; A S S E M B L E R - S W I T C H E S
;===============================================================================

OPTIMIZE        = 0                     ; enable some possible optimizations
FILL_OPT        = 1                     ; fill optimized bytes with NOPs
SCREENSAVER     = 1                     ; compile with screensaver code
TRAINER         = 0                     ; enable training mode
NTSC            = 1                     ; compile for NTSC
  IF NTSC
FRAMERATE       = 60
STARTTIME       = $20
  ELSE
FRAMERATE       = 60                    ; 50    use these values to..
```

```
STARTTIME       = $20                   ; $24   ..adjust the PAL timer
  ENDIF


;===============================================================================
; C O N S T A N T S
;===============================================================================

; initial value for the random number generator:
RAND_SEED       = $c4                   ; defines the start scene of the game

; color constants:
  IF NTSC
BROWN           = $12
YELLOW          = $1e
ORANGE          = $3e
RED             = $48
GREEN           = $d6
BLUE            = $a4
YELLOW_GREEN    = $c8
PINK            = $4a
  ELSE
BROWN           = $22
YELLOW          = $2e
ORANGE          = $4e
RED             = $68
GREEN           = $56
BLUE            = $b4
YELLOW_GREEN    = $78
PINK            = $4a
  ENDIF
BLACK           = $00
GREY            = $06
WHITE           = $0e
DARK_GREEN      = GREEN  - $04
DARK_RED        = RED    - $06

; x constants:
SCREENWIDTH     = 160
XMIN_HARRY      = 8                     ; minimal position before next left scene
XMAX_HARRY      = 148                   ; maximal position before next right scene

; y-size constants:
HARRY_H         = 22                    ; height of Harry
OBJECT_H        = 16                    ; maximum object height
DIGIT_H         = 8                     ; height of the score and time digits
COPYRIGHT_H     = 16                    ; height of copyright message

; some defined y-positions of Harry:
JUNGLE_GROUND   = 32
UNDER_GROUND    = 86

; positions of Harry at ladder:
LADDER_TOP      = 11
LADDER_BOTTOM   = 22

; lenght of a jump:
JUMP_LEN        = 32
```

```
; Harry pattern ids:
ID_KNEEING      = 0
ID_RUNNING4     = 4                     ; 0..3 are running ids too
ID_STANDING     = 5
ID_SWINGING     = 6
ID_CLIMBING     = 7

; objectType ids:
ID_STATIONARY   = 4                     ; stationary logs (0..5 are log types)
ID_FIRE         = 6
ID_COBRA        = 7
ID_TREASURES    = 8                     ; 8..11 are treasures
ID_NOTHING      = 12

; sceneType constants:
HOLE1_SCENE     = 0
HOLE3_SCENE     = 1
CROCO_SCENE     = 4
TREASURE_SCENE  = 5

; flags for ladder:
NOLADDER        = %00000000
WITHLADDER      = %11111111

; flags for pit color:
BLUEPIT         = %00000000
BLACKPIT        = %10000000

; offsets in SoundTab for tunes:
SOUND_JUMP      = $20                   ; Harry is jumping
SOUND_TREASURE  = $25                   ; Harry is collecting a treasure
SOUND_DEAD      = $31                   ; Harry is killed
SOUND_FALLING   = $53                   ; Harry is falling into a hole

; values for NUSIZx:
ONE_COPY        = %000
TWO_COPIES      = %001
TWO_WIDE_COPIES = %010
THREE_COPIES    = %011
DOUBLE_SIZE     = %101
THREE_MED_COPIES = %110
QUAD_SIZE       = %111

; mask for SWCHB:
BW_MASK         = %1000                 ; black and white bit

; SWCHA joystick bits:
MOVE_RIGHT      = %0111
MOVE_LEFT       = %1011
MOVE_DOWN       = %1101
MOVE_UP         = %1110
NO_MOVE         = %1111
JOY_RIGHT       = ~MOVE_RIGHT & NO_MOVE
JOY_LEFT        = ~MOVE_LEFT  & NO_MOVE
JOY_DOWN        = ~MOVE_DOWN  & NO_MOVE
JOY_UP          = ~MOVE_UP    & NO_MOVE
JOY_HORZ        = JOY_RIGHT|JOY_LEFT
```

```
JOY_VERT        = JOY_DOWN |JOY_UP

; values for ENAxy:
DISABLE         = %00
ENABLE          = %10                   ; value for enabling a missile

; values for REFPx:
NOREFLECT       = %0000
REFLECT         = %1000


;===============================================================================
; Z P - V A R I A B L E S
;===============================================================================

    SEG.U   Variables
    ORG     $80

livesPat            .byte   ;           number of lives, stored as displayed
pattern ($a0 = 3, $80 = 2, $00 = 1)
random              .byte   ;           all scenes are generated randomly with
this
random2             .byte   ;           used for random object animation
joystick            .byte   ;           stores joystick directions
fireButton          .byte   ;           stores fire button state
hitLiana            .byte   ;           Harry collided with liana? (bit 6 = 1 ->
yes)
cxHarry             .byte   ;           Harry's collisions (stored but _never_
read!)
  IF SCREENSAVER
SS_XOR              .byte   ;           change colors in screensaver mode
(0/$01..$ff)
SS_Mask             .byte   ;           darker colors in screensaver mode
($ff/$f7)
  ELSE
dummy               .word
  ENDIF
colorLst            ds 9    ;           some (mostly constant!?) colors
lianaBottom         .byte   ;           bottom row of liana
objectType          .byte   ;           type of the objects on the ground (hazards
& treasures)
sceneType           .byte   ;           type of the scene (0..7
HMFineLst           ds 3    ;           fine positioning value for: Harry, ground-
object, underground-object
HMCoarseLst         ds 3    ;           coars positioning value for: Harry,
ground-object, underground-object
posLeftBranch       .byte   ;           values for positioning left branch
graphics
posRightBranch      .byte   ;           values for positioning right branch
graphics
ladderFlag          .byte   ;           0 = no ladder, $ff = with ladder
noGameScroll        .byte   ;           0 = game is running
PF2QuickSand        .byte   ;           PF2 data for top quicksand row
PF2Lst              ds 7    ;           copied pit pattern data
objColLst           ds 7    ;           copied object colors
objPatLst           ds 7    ;           copied object patterns
harryPatPtr         .word   ; = $b5     pointer to Pitfall Harry patterns
objPatPtr           .word   ;           pointer to object (hazards, treasure)
```

```
patterns
harryColPtr         .word   ;           pointer to Pitfall Harry colors
objColPtr           .word   ;           pointer to object (hazards, treasure)
colors
wallPatPtr          .word   ;           pointer to wall patterns
wallColPtr          .word   ;           pointer to wall colors
undrPatPtr          .word   ;           pointer to underground object (wall,
scorpion) patterns
undrColPtr          .word   ;           pointer to underground object (wall,
scorpion) colors
digitPtr            ds.w 6  ;           pointers for score display
  IF SCREENSAVER
SS_Delay            .byte   ; = $d1
SS_DelayLo          .byte
  ENDIF
frameCnt            .byte   ;           frame counter (increased every frame)
nusize1             .byte   ;           number of ground-objects
scoreHi             .byte   ;           3 BCD score bytes
scoreMed            .byte
scoreLo             .byte
timerHi             .byte   ;           2 BCD timer bytes
timerMed            .byte
timerLo             .byte   ;           decease timer every 60th frame
hmblSum             .byte   ;           used to generate liana line
hmblAdd             .byte   ;           depends on the liana angle
hmblDir             .byte   ;           move liana +/-1
lianaPosHi          .byte   ;           high x-position of liana bottom
lianaPosLo          .byte   ;           low x-position of liana bottom
soundIdx            .byte   ;           index of sound-table (0 = no sound)
xPosHarry           .byte   ;           x-position of Pitfall Harry
xPosObject          .byte   ;           x-position of hazards & treasures
xPosScorpion        .byte   ;           x-position of the scorpion (and the wall)
patIdHarry          .byte   ;           id of the animation for Harry
reflectHarry        .byte   ;           reflect Harry graphics
reflectScorpion     .byte   ;           reflect scorpion graphics
jumpIndex           .byte   ;           index of jump-table (0..32)
oldJoystick         .byte   ;           saved old joystick direction
yPosHarry           .byte   ;           y-position of Pitfall Harry
atLiana             .byte   ;           Harry at liana? (0 = no, -1 = yes)
treePat             .byte   ;           id of the leaves pattern (0..3)
climbPos            .byte   ;           position of Harry at ladder (0/11..22)
treasureBits        ds 4    ;           remember which treasures haven't been
found
treasureCnt         .byte   ; = $f1     number of remaining treasures-1
patOfsHarry         .byte   ;           pattern offset (5 while kneeing, 0 else)
soundDelay          .byte   ;           play a new note every 4th frame
xPosQuickSand       .byte   ;           border of quicksand
jumpMode            .byte   ; = $f5     similar to jumpIndex (JTZ: superfluous?)
temp1               .byte
temp2               .byte
temp3               .byte


;===============================================================================
; M A C R O S
;===============================================================================

  MAC FILL_NOP
```

```
    IF FILL_OPT
      REPEAT {1}
         NOP
      REPEND
    ENDIF
  ENDM


;===============================================================================
; R O M - C O D E (Part 1)
;===============================================================================

    SEG     Code
    ORG     $f000, 0

START:
    sei                     ; 2
    cld                     ; 2
    ldx    #$00             ; 2
Reset:
    lda    #$00             ; 2
.loopClear:
    sta    $00,x            ; 4
    txs                     ; 2
    inx                     ; 2
    bne    .loopClear       ; 2

    jsr    InitGame         ; 6

MainLoop:
    ldx    #8               ; 2
.loopColors:
    lda    ColorTab,x       ; 4
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 4
  ENDIF
    sta    colorLst,x       ; 4                 store color in list
    cpx    #4               ; 2
    bcs    .skipTIA         ; 2
    sta    COLUP0,x         ; 4                 store color in TIA too
.skipTIA:
    dex                     ; 2
    bpl    .loopColors      ; 2

; process underground objects (scorpion, wall):
; Only one object at a time is visible, but two different
; pointer sets are used, because the wall is much taller than
; the scorpion.
; JTZ: two color pointers aren't necessary!
    ldy    #<ScorpionColor  ; 2
    ldx    #<Scorpion0      ; 2
    lda    xPosScorpion     ; 3
    lsr                     ; 2                 animate scorpion
    bcc    .scorpion0       ; 2
    ldx    #<Scorpion1      ; 2
```

```
.scorpion0:
    lda    #<Nothing        ; 2
    sta    wallPatPtr       ; 3                clear ladder data pointer

    lda    ladderFlag       ; 3                ladder in scene
    beq    .noLadder        ; 2                 no, skip
    ldy    #<WallColor      ; 2                 yes,..
    ldx    #<Wall           ; 2
    stx    wallPatPtr       ; 3                ..load pointers at ladder data
    sty    wallColPtr       ; 3
.noLadder:
    stx    undrPatPtr       ; 3                set scorpion or ladder pointers
    sty    undrColPtr       ; 3

; calculate pits, quicksand etc.:
    ldx    sceneType        ; 3
    lda    LadderTab,x      ; 4                blue swamp?
    bpl    .noPit           ; 2                 yes, skip
    lda    colorLst+4       ; 3                 no, black tar pit
    sta    colorLst+8       ; 3
.noPit:
    ldy    #0               ; 2                disable quicksand
    lda    GroundTypeTab,x  ; 4
    bpl    .noQuickSand     ; 2
    lda    yPosHarry        ; 3
    cmp    #55              ; 2                Harry in underground?
    bcs    .doQuickSand     ; 2                 yes, animate quicksand
    cmp    #JUNGLE_GROUND+1 ; 2                stop quicksand animation when..
    bcs    .stopQuickSand   ; 2                ..Harry is falling into the pit
.doQuickSand:
    lda    noGameScroll     ; 3                game running?
    bne    .stopQuickSand   ; 2                 no, skip
    lda    frameCnt         ; 3
    lsr                     ; 2
    lsr                     ; 2
    pha                     ; 3
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2
    tax                     ; 2                x = framecount / 64
    pla                     ; 4                only bits 2..5 of framecounter
used
    and    QuickSandTab+2,x ; 4                calculate size of the quicksand
pit
    eor    QuickSandTab,x   ; 4
    pha                     ; 3
    tay                     ; 2
    lda    QuickSandSize,y  ; 4
    tay                     ; 2
    pla                     ; 4
    clc                     ; 2
    adc    #16              ; 2
.noQuickSand:
    clc                     ; 2
    sty    xPosQuickSand    ; 3
    adc    #6               ; 2
    tay                     ; 2
```

```
    ldx    #6               ; 2
    lda    ladderFlag       ; 3                 no swamp etc. when ladder
    eor    #$ff             ; 2
    sta    temp1            ; 3
.loopPF2Lst:
    lda    PF2PatTab,y      ; 4
    sta    PF2Lst,x         ; 4
    ora    temp1            ; 3
    sta    PF2QuickSand     ; 3
    dey                     ; 2
    dex                     ; 2
    bpl    .loopPF2Lst      ; 2
.stopQuickSand:

; calculate x-positioning values:
    ldx    #2               ; 2
    lda    #0               ; 2
  IF SCREENSAVER
    ldy    SS_Delay         ; 3                 game running?
    bmi    .skipHarryPos    ; 2                  no, don't draw Harry
  ELSE
    ldy    noGameScroll     ;                   TODO: bugfix, wall isn't drawn
    bne    .skipHarryPos    ; 2                  no, don't draw Harry
  ENDIF

.loopPos:
    lda    xPosHarry,x      ; 4
.skipHarryPos:
    jsr    CalcPosX         ; 6
    sta    HMFineLst,x      ; 4
    sty    HMCoarseLst,x    ; 4
    stx    hmblSum          ; 3                 -> hmblSum = 0
    dex                     ; 2
    bpl    .loopPos         ; 2

; load branches x-positioning values:
    ldx    treePat          ; 3
    lda    BranchPosLTab,x  ; 4
    sta    posLeftBranch    ; 3
    lda    BranchPosRTab,x  ; 4
    sta    posRightBranch   ; 3

; copy bottom object data:
    ldy    #14              ; 2
    ldx    #6               ; 2
.loopObjLst:
    lda    (objColPtr),y    ; 5
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 4
  ENDIF
    sta    objColLst,x      ; 4
    lda    (objPatPtr),y    ; 5
    sta    objPatLst,x      ; 4
    dey                     ; 2
    dex                     ; 2
```

```
    bpl    .loopObjLst      ; 2
.waitTim:
    lda    INTIM            ; 4
    bne    .waitTim         ; 2
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    sta    VBLANK           ; 3
    sta    CXCLR            ; 3
    sta    temp3            ; 3                don't show anything before score
    jsr    ShowDigits       ; 6                draw score

; set digitPtrs for timer:
    ldx    #timerHi-scoreHi ; 2
    ldy    #2               ; 2                minutes
    jsr    BCD2DigitPtrs    ; 6
    inx                     ; 2                seconds
    ldy    #8               ; 2
    jsr    BCD2DigitPtrs    ; 6
    lda    livesPat         ; 3                show lives before time
    sta    temp3            ; 3
    lda    #<Space          ; 2
    sta    digitPtr         ; 3
    ldy    digitPtr+2       ; 3
    bne    .noSpace         ; 2                replaced leading zero in timer
with space
    sta    digitPtr+2       ; 3
.noSpace:
    lda    #<DoublePoint    ; 2
    sta    digitPtr+6       ; 3
    jsr    ShowDigits       ; 6                draw lives and timer
    sta    WSYNC            ; 3
;--------------------------------------
; Here starts the main kernel. Actually there are nine(!)
; specialized kernel loops. Together with extra code before and
; after the loops, this makes the kernel very huge [~900 bytes).

; draw branches at top of the logs:
    sta    HMOVE            ; 3
    lda    #$00             ; 2
    sta    VDELP1           ; 3                disable vertical delay for object
    sta    GRP0             ; 3
    sta    GRP1             ; 3
    lda    colorLst+5       ; 3                branches color (always BROWN-2)
    sta    COLUP0           ; 3
    sta    COLUP1           ; 3
    lda    #TWO_WIDE_COPIES ; 2                draw four branches
    sta    NUSIZ0           ; 3
    sta    NUSIZ1           ; 3
    lda    posLeftBranch    ; 3
    and    #$0f             ; 2
    tax                     ; 2                x = coarse x-positioning value of
left branch
    lda    posRightBranch   ; 3
    and    #$0f             ; 2
    tay                     ; 2                y = coarse x-positioning value of
right branch
```

```
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    nop                     ; 2
.waitPos0:
    dex                     ; 2
    bpl    .waitPos0        ; 2
    sta    RESP0            ; 3
    lda    posLeftBranch    ; 3
    sta    HMP0             ; 3
    lda    posRightBranch   ; 3
    sta    HMP1             ; 3
.waitPos1:
    dey                     ; 2
    bpl    .waitPos1        ; 2
    sta    RESP1            ; 3
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    lda    #%101            ; 2                enable playfield priority, now..
    sta    CTRLPF           ; 3                ..the leaves overlap the branches
    ldy    #31              ; 2
    lda    treePat          ; 3
    asl                     ; 2
    asl                     ; 2
    tax                     ; 2

; Kernel 1 (31 lines): draw liana, branches and bottom of leaves:
.loopBranches:
    clc                     ; 2
    lda    hmblSum          ; 3
    adc    hmblAdd          ; 3
    sta    hmblSum          ; 3
    sta    HMCLR            ; 3
    bcc    .noMove0         ; 2
    lda    hmblDir          ; 3
    sta    HMBL             ; 3
.noMove0:
    lda    #0               ; 2
    cpy    #9               ; 2                draw branches in lower 9 lines
    bcs    .noBranch        ; 2
    tya                     ; 2
    lsr                     ; 2
    lda    BranchTab,y      ; 4
.noBranch:
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    sta    GRP0             ; 3
    sta    GRP1             ; 3
    bcs    .noChangePF      ; 2                two line resolution for leaves
    lda    PFLeavesTab,x    ; 4                x = 0..3
    inx                     ; 2
    sta    PF0              ; 3
    sta    PF1              ; 3
    sta    PF2              ; 3
.noChangePF:
    dey                     ; 2
```

```
    bne    .loopBranches    ; 2

; prepare Kernel 2: draw liana, disable branches, draw logs:
    ldx    treePat          ; 3
    clc                     ; 2
    lda    hmblSum          ; 3
    adc    hmblAdd          ; 3
    sta    hmblSum          ; 3
    bcc    .noMove1         ; 2
    ldy    hmblDir          ; 3
.noMove1:
    sty    HMBL             ; 3
    lda    #%001            ; 2
    sta    CTRLPF           ; 3                 disable playfield priority
    lda    PF1LogTab,x      ; 4
    ldy    PF2LogTab,x      ; 4
    ldx    nusize1          ; 3
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    sta    PF1              ; 3                 draw outer logs
    lda    colorLst+5       ; 3                 always BROWN-2
    sta    COLUPF           ; 3
    lda    #0               ; 2
    sta    GRP0             ; 3
    sta    GRP1             ; 3
    sta    NUSIZ0           ; 3
    sta    PF0              ; 3
    sty    PF2              ; 3                 draw inner logs
    stx    NUSIZ1           ; 3

; Kernel 2 (4 lines): draw liana, position Harry and other object:
    ldx    #1               ; 2
.loopLianaPos:
    clc                     ; 2
    lda    hmblSum          ; 3
    adc    hmblAdd          ; 3
    sta    hmblSum          ; 3
    lda    #$00             ; 2
    bcc    .noMove2         ; 2
    lda    hmblDir          ; 3
.noMove2:
    sta    HMBL             ; 3
    clc                     ; 2                 precalc liana for next line
    lda    hmblSum          ; 3
    adc    hmblAdd          ; 3
    sta    hmblSum          ; 3
    lda    #$00             ; 2
    bcc    .noMove3         ; 2
    lda    hmblDir          ; 3
.noMove3:
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    ldy    #0               ; 2                 do the coarse positions
    sty    temp1,x          ; 4
    ldy    HMCoarseLst,x    ; 4                 position at the very left?
    bne    .waitPos         ; 2                  no, skip
```

```
    ldy    #$60             ; 2                  yes, use special code
    sty    temp1,x          ; 4
    sta    RESP0,x          ; 4
    sta    HMBL             ; 3
    bne    .endPos0         ; 3

.waitPos:
    dey                     ; 2                 "normal" position
    bne    .waitPos         ; 2
    sta.w  HMBL             ; 4
    sta    RESP0,x          ; 4
.endPos0:
    sta    WSYNC            ; 3
;---------------------------------------
    sta    HMOVE            ; 3
    dex                     ; 2
    bpl    .loopLianaPos    ; 2

    jsr    DrawLiana        ;31/33
    lda    HMFineLst        ; 3
    sta    HMP0             ; 3                 do the fine positions
    lda    HMFineLst+1      ; 3
    sta    HMP1             ; 3
    sta    WSYNC            ; 3
;---------------------------------------
    sta    HMOVE            ; 3
    jsr    DrawLiana        ;31/33
    lda    temp1            ; 3
    sta    HMP0             ; 3
    lda    temp2            ; 3
    sta    HMP1             ; 3
    lda    yPosHarry        ; 3                 calculate offset for Harry's
pattern
    clc                     ; 2
    adc    patOfsHarry      ; 3
    adc    #21              ; 2
    tay                     ; 2
    lda    reflectHarry     ; 3
    sta    REFP0            ; 3
    sta    WSYNC            ; 3
;---------------------------------------
    sta    HMOVE            ; 3
    sta    CXCLR            ; 3
    ldx    #20              ; 2
    stx    VDELP0           ; 3                 disable vertical delay for Harry

; Kernel 3 (21 lines): draw liana and Harry:
.loopLianaHarry:
    clc                     ; 2
    lda    hmblSum          ; 3
    adc    hmblAdd          ; 3
    sta    hmblSum          ; 3
    lda    #$00             ; 2
    sta    HMCLR            ; 3
    bcc    .noMove4         ; 2
    lda    hmblDir          ; 3
.noMove4:
    sta    HMBL             ; 3
```

```
    jsr    DrawHarry        ;27/37
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    sta    COLUP0           ; 3
    dex                     ; 2
    bpl    .loopLianaHarry  ; 2

    stx    VDELP0           ; 3                 enable vertical delay for Harry
    inx                     ; 2
    stx    GRP1             ; 3
    beq    .endLiana        ; 3

.skipHarry:
    lda    #0               ; 2
    sta    GRP0             ; 3
    beq    .contendLiana    ; 3

.endLiana:
    ldx    #23              ; 2
; Kernel 4 (24 lines): draw end of liana, draw Harry:
.loopEndLiana:
    clc                     ; 2
    lda    hmblSum          ; 3
    adc    hmblAdd          ; 3
    sta    hmblSum          ; 3
    lda    #$00             ; 2
    bcc    .noMove5         ; 2
    lda    hmblDir          ; 3
.noMove5:
    sta    HMBL             ; 3
    dey                     ; 2
    cpy    #HARRY_H         ; 2
    bcs    .skipHarry       ; 2
    lda    (harryPatPtr),y  ; 5
    sta    GRP0             ; 3
    lda    (harryColPtr),y  ; 5
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 4
  ENDIF
.contendLiana:
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    sta    COLUP0           ; 3
    lda    #DISABLE         ; 2
    cpx    lianaBottom      ; 3                 bottom of liana reached?
    bcs    .skipDisable     ; 2                  no, skip
    sta    ENABL            ; 3                  yes, disable liana
.skipDisable:
    sta    GRP1             ; 3
    dex                     ; 2
    bpl    .loopEndLiana    ; 2

    jsr    DrawHarry        ;27/37
```

```
    ldx    CXP0FB-$30       ; 3
    stx    hitLiana         ; 3
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    sta    COLUP0           ; 3
    lda    colorLst+7       ; 3
    sta    COLUPF           ; 3
    ldx    #$ff             ; 2                 draw the jungle ground
    stx    PF0              ; 3
    stx    PF1              ; 3
    stx    PF2              ; 3
    lda    colorLst+8       ; 3
    sta    COLUBK           ; 3
    inx                     ; 2
    stx    GRP1             ; 3

    ldx    #6               ; 2
; Kernel 5: draw Harry, holes, top of object on the ground:
.loopGround:
    jsr    DrawHarry        ;27/37
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    sta    COLUP0           ; 3
    lda    objColLst,x      ; 4
    sta    COLUP1           ; 3
    lda    objPatLst,x      ; 4                 draw object (crocodiles, logs,
snake...)
    sta    GRP1             ; 3
    lda    PF2Lst,x         ; 4                 draw pits
    sta    PF2              ; 3
    dex                     ; 2
    bpl    .loopGround      ; 2=31/32

    tya                     ; 2                 calculate and save..
    sec                     ; 2                 ..Harry's pattern..
    sbc    #8               ; 2                 ..offset for kernel 7
    sta    temp1            ; 3

    ldx    #0               ; 2
    ldy    #7               ; 2
; Kernel 6 (8 lines): draw bottom of object on the ground, holes in the ground:
.loopHoles:
    lda    #0               ; 2
    sta    GRP0             ; 3
    lda    (objColPtr),y    ; 5
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 4
  ENDIF
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    sta    COLUP1           ; 3
    lda    (objPatPtr),y    ; 5
```

```
    sta    GRP1             ; 3
    dey                     ; 2
    bmi    .exitHoles       ; 2                exit loop here
    lda    PF2Lst,x         ; 4
    sta    PF2              ; 3
    inx                     ; 2
    bne    .loopHoles       ; 3                loop always

.exitHoles:
    lda    #0               ; 2
    sta    GRP0             ; 3                clear Harry again (JTZ:
superfluous)
    ldx    HMCoarseLst+2    ; 3
    bne    .notZero         ; 2
    lda    #$60             ; 2                special HMOV when scorpion is at
the very left
.notZero:
    sta    temp3            ; 3

; check Harry's collisions (JTZ: superfluous code!)
    lda    CXPPMM-$30       ; 3                Harry collided with other objects?
    asl                     ; 2
    lda    CXP0FB-$30       ; 3                Harry collided with playfield?
    ror                     ; 2
    sta    cxHarry          ; 3                store here (this variable isn't
used somewhere else!)

; prepare some underground data:
    lda    reflectScorpion  ; 3                set player 1 reflection
    sta    REFP1            ; 3
    lda    ladderFlag       ; 3                calculate playfield reflection
    and    #%000100         ; 2
    eor    #%100101         ; 2                ball is 4 clocks wide (ladder)
    tax                     ; 2
    ldy    colorLst+6       ; 3                underground color (always BROWN+2)
    lda    colorLst+4       ; 3                hole, blackground and tar pit
color
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    jmp    ContKernel       ; 3

ShowDigits SUBROUTINE
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    lda    colorLst         ; 3
    sta    COLUP0           ; 3
    sta    COLUP1           ; 3
    ldy    #0               ; 2
    sty    REFP0            ; 3
    sty    REFP1            ; 3
    ldx    #$10|THREE_COPIES; 2
    stx    NUSIZ0           ; 3
    sta    RESP0            ; 3
    sta    RESP1            ; 3
    stx    HMP1             ; 3
    sta    WSYNC            ; 3
```

```
;--------------------------------------
    sta    HMOVE            ; 3
    stx    NUSIZ1           ; 3
    iny                     ; 2
    sty    CTRLPF           ; 3                  enable playfield reflection
    lda    #DIGIT_H-1       ; 2
    sta    VDELP0           ; 3
    sta    VDELP1           ; 3
    sta    temp2            ; 3
    sta    HMCLR            ; 3
    jsr    SkipIny          ;22                  just waste 22 cycles
    lda    temp3            ; 3                  just waste three cycles
.loopDigits:
    ldy    temp2            ; 3
    lda    (digitPtr+10),y  ; 5
    sta    temp1            ; 3
    lda    (digitPtr+8),y   ; 5
    tax                     ; 2
    lda    (digitPtr),y     ; 5
    ora    temp3            ; 3                  show lives when drawing time
    sta    HMOVE            ; 3                  produce HMOVE blanks
    sta    GRP0             ; 3
    lda    (digitPtr+2),y   ; 5
    sta    GRP1             ; 3
    lda    (digitPtr+4),y   ; 5
    sta    GRP0             ; 3
    lda    (digitPtr+6),y   ; 5
    ldy    temp1            ; 3
    sta    GRP1             ; 3
    stx    GRP0             ; 3
    sty    GRP1             ; 3
    sta    GRP0             ; 3
    dec    temp2            ; 5
    bpl    .loopDigits      ; 2

    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    lda    #0               ; 2
    sta    GRP0             ; 3
    sta    GRP1             ; 3
    sta    GRP0             ; 3
    rts                     ; 6

DrawHarry SUBROUTINE
; called from kernel:
    dey                     ; 2
    cpy    #HARRY_H         ; 2
    bcs    .skipDraw        ; 2
    lda    (harryPatPtr),y  ; 5
    sta    GRP0             ; 3
    lda    (harryColPtr),y  ; 5
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 3
  ENDIF
```

```
.exitDraw:
    rts                     ; 6 = 21/31

  IF SCREENSAVER = 0
; do some missing nops:
    FILL_NOP 2
  ENDIF

.skipDraw:                  ; 7
    lda    #0               ; 2
    sta    GRP0             ; 3
    beq    .exitDraw        ; 3

CalcPosX SUBROUTINE
; calculate coarse and fine x-positioning values:
    tay                     ; 2
    iny                     ; 2
    tya                     ; 2
    and    #$0f             ; 2
    sta    temp1            ; 3
    tya                     ; 2
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2
    tay                     ; 2
    clc                     ; 2
    adc    temp1            ; 3
    cmp    #$0f             ; 2
    bcc    SkipIny          ; 2
    sbc    #$0f             ; 2
    iny                     ; 2
SkipIny:
    eor    #$07             ; 2
    asl                     ; 2
    asl                     ; 2
    asl                     ; 2
    asl                     ; 2
    rts                     ; 6

DrawLiana SUBROUTINE
    clc                     ; 2
    lda    hmblSum          ; 3
    adc    hmblAdd          ; 3
    sta    hmblSum          ; 3
    lda    #$00             ; 2
    bcc    .noMove6         ; 2
    lda    hmblDir          ; 3
.noMove6:
    sta    HMBL             ; 3
    rts                     ; 6 = 25/27

BCD2DigitPtrs SUBROUTINE
    lda    scoreHi,x        ; 4
    and    #$f0             ; 2
    lsr                     ; 2
    sta    digitPtr,y       ; 5
    lda    scoreHi,x        ; 4
```

```
    and    #$0f             ; 2
    asl                     ; 2
    asl                     ; 2
    asl                     ; 2
    sta    digitPtr+2,y     ; 5
    sta    WSYNC            ; 3
    sta    HMOVE            ; 3
    rts                     ; 6

DecScoreLo SUBROUTINE
    lda    #$07             ; 2
    sta    AUDC1            ; 3
    lda    #$99             ; 2              decrease scoreLo by 1
DecScoreHi:                 ;                decrease scoreHi by 1
    sed                     ; 2
    clc                     ; 2
    adc    scoreLo          ; 3
    sta    scoreLo          ; 3
    lda    scoreMed         ; 3
    sbc    #$00             ; 2
    sta    scoreMed         ; 3
    lda    scoreHi          ; 3
    sbc    #$00             ; 2
    bcs    .notZero         ; 2
    lda    #$00             ; 2             limit score at zero
    sta    scoreMed         ; 3
    sta    scoreLo          ; 3
.notZero:
    sta    scoreHi          ; 3
    cld                     ; 2
    rts                     ; 6

PFLeavesTab:
    .byte %11111111 ; |XXXXXXXX|
    .byte %11001111 ; |XX  XXXX|
    .byte %10000011 ; |X     XX|
    .byte %00000001 ; |       X|

    .byte %01111111 ; | XXXXXXX|
    .byte %00111101 ; |  XXXX X|
    .byte %00011000 ; |   XX   |
    .byte %00000000 ; |        |

    .byte %11111111 ; |XXXXXXXX|
    .byte %11111110 ; |XXXXXXX |
    .byte %10111100 ; |X XXXX  |
    .byte %00011000 ; |   XX   |

    .byte %11111110 ; |XXXXXXX |
    .byte %11111100 ; |XXXXXX  |
    .byte %01111000 ; | XXXX   |
    .byte %00110000 ; |  XX    |

Kernel2 SUBROUTINE
.skipHarry1:
    lda    #0               ; 2
    sta    GRP0             ; 3
    beq    .conrHarry       ; 3
```

```
ContKernel:
    sta    COLUBK           ; 3
    sty    COLUPF           ; 3
    lda    #0               ; 2
    sta    GRP1             ; 3
    lda    #$ff             ; 2
    sta    PF1              ; 3
    lda    PF2QuickSand     ; 3
    sta    PF2              ; 3
    stx    CTRLPF           ; 3
    ldy    temp1            ; 3
    lda    #$90             ; 2
    sta.w  HMBL             ; 4
    cpy    #HARRY_H         ; 2
    sta    RESBL            ; 3
    bcs    .skipHarry1      ; 2?+1
    lda    (harryPatPtr),y  ; 5
    sta    GRP0             ; 3
    lda    (harryColPtr),y  ; 5
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 4
  ENDIF
.conrHarry:
    ldx    HMCoarseLst+2    ; 3
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    sta    COLUP0           ; 3
    beq    .wait1           ; 2?
.wait1:
    beq    .wait2           ; 2?
.wait2:
    lda    #0               ; 2
    sta    GRP1             ; 3
.loopWait:
    dex                     ; 2
    bpl    .loopWait        ; 2?
    sta.w  RESP1            ; 4
    sta    HMCLR            ; 3
    sta    WSYNC            ; 3
;--------------------------------------
    sta    HMOVE            ; 3
    dey                     ; 2
    cpy    #HARRY_H         ; 2
    bcs    .skipHarry2      ; 2?
    lda    (harryColPtr),y  ; 5
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 3
  ENDIF
    sta    COLUP0           ; 3
    lda    (harryPatPtr),y  ; 5
```

```
    sta    GRP0             ; 3
.skipHarry2:
    lda    #0               ; 2
    sta    GRP1             ; 3
    lda    HMFineLst+2      ; 3
    sta    HMP1             ; 3                 position scorpion or wall
    ldx    #11              ; 2
    dey                     ; 2
; Kernel 7 (12 lines): draw top of ladder, draw Harry:
.loopLadderTop:
    dey                     ; 2
    cpy    #HARRY_H         ; 2
    bcs    .skipHarry3      ; 2
    lda    (harryPatPtr),y  ; 5
    sta    GRP0             ; 3
    lda    (harryColPtr),y  ; 5
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 4
  ENDIF
.contHarry3:
    sta    WSYNC            ; 3
;---------------------------------------
    sta    HMOVE            ; 3
    sta    COLUP0           ; 3
    lda    #0               ; 2
    sta    GRP1             ; 3
    lda    LadderTab,x      ; 4
    and    ladderFlag       ; 3
    sta    ENABL            ; 3
    dex                     ; 2
    bmi    .exitLadderTop   ; 2                 exit loop
    lda    temp3            ; 3
    sta    HMCLR            ; 3
    sta    HMP1             ; 3                 position scorpion at the very left
    lda    #15              ; 2                 clear hmove value, prepare height
of a later loop
    sta    temp3            ; 3
    bne    .loopLadderTop   ; 3                 loop always

.skipHarry3:
    lda    #0               ; 2
    sta    GRP0             ; 3
    beq    .contHarry3      ; 3

.skipHarry4:
    lda    #0               ; 2
    sta    GRP0             ; 3
    beq    .contHarry4      ; 3

.exitLadderTop:
    dey                     ; 2
    sty    temp1            ; 3
    cpy    #HARRY_H         ; 2
    bcs    .skipHarry4      ; 2
    lda    (harryPatPtr),y  ; 5
```

```
    sta    GRP0             ; 3
    lda    (harryColPtr),y  ; 5
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 4
  ENDIF
.contHarry4:
    ldy    #15              ; 2
    sta    temp2            ; 3
    lda    (wallPatPtr),y   ; 5
    ldx    #ONE_COPY        ; 2                 = 0
    stx    NUSIZ1           ; 3
    sta    WSYNC            ; 3
;---------------------------------------
    sta    HMOVE            ; 3
    sta    GRP1             ; 3
    lda    temp2            ; 3
    sta    COLUP0           ; 3
    stx    PF0              ; 3                 clear playfield
    lda    #DARK_RED        ; 2                 wall color
  IF SCREENSAVER
    and    SS_Mask          ; 3
  ELSE
    FILL_NOP 2
  ENDIF
    sta    COLUP1           ; 3
    stx    PF1              ; 3
    stx    PF2              ; 3
    lda    ladderFlag       ; 3
    sta    ENABL            ; 3
    dey                     ; 2
    sty    temp2            ; 3
    ldx    temp1            ; 3
; Kernel 8 (15 lines): draw Harry, ladder and wall:
.loopLadder:
    dex                     ; 2
    txa                     ; 2
    tay                     ; 2
    cpy    #HARRY_H         ; 2
    bcs    .skipHarry5      ; 2*+1
    lda    (harryPatPtr),y  ; 5
    sta    GRP0             ; 3
    lda    (harryColPtr),y  ; 5
.contHarry5:
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    bit    $00
    bit    $00
  ENDIF
    ldy    temp2            ; 3
    sta    HMOVE            ; 3
    sta    COLUP0           ; 3
    lda    (wallPatPtr),y   ; 5
    sta    GRP1             ; 3
```

```
    lda    (wallColPtr),y   ; 5
  IF SCREENSAVER
    and    SS_Mask          ; 3
  ELSE
    bit    $00
  ENDIF
    sta    COLUP1           ; 3
    lda    LadderTab,y      ; 4
    and    ladderFlag       ; 3
    sta    ENABL            ; 3
    dec    temp2            ; 5
    bpl    .loopLadder      ; 2@+1
    nop                     ; 2
; Kernel 9 (16 lines): draw Harry, scorpion or the bottom of wall and ladder:
.loopUnderground:
    dex                     ; 2
    txa                     ; 2
    tay                     ; 2
    cpy    #HARRY_H         ; 2
    bcs    .skipHarry6      ; 2@
    lda    (harryPatPtr),y  ; 5
    sta    GRP0             ; 3
    lda    (harryColPtr),y  ; 5
.contHarry6:
  IF SCREENSAVER
    eor    SS_XOR           ; 3
    and    SS_Mask          ; 3
  ELSE
    bit    $00
    bit    $00
  ENDIF
    ldy    temp3            ; 3
    sta    HMOVE            ; 3
    sta    COLUP0           ; 3
    lda    (undrPatPtr),y   ; 5
    sta    GRP1             ; 3
    lda    (undrColPtr),y   ; 5
  IF SCREENSAVER
    and    SS_Mask          ; 3
  ELSE
    bit    $00
  ENDIF
    sta    COLUP1           ; 3
    lda    LadderTab,y      ; 4
    and    ladderFlag       ; 3
    sta.w  ENABL            ; 4
    dec    temp3            ; 5
    bpl    .loopUnderground ; 2@
    bmi    .exitKernel      ; 3

.skipHarry5:
    lda    #0               ; 2
    sta    GRP0             ; 3
    nop                     ; 2
    beq    .contHarry5      ; 4                 page crossed!

.skipHarry6:
    lda    #0               ; 2
```

```
    sta    GRP0             ; 3
    nop                     ; 2
    nop                     ; 2
    beq    .contHarry6      ; 3

.exitKernel:
    ldx    #$ff             ; 2
    sta    WSYNC            ; 3
;---------------------------------------
    sta    HMOVE            ; 3
    stx    PF0              ; 3                fill playfield registers
    stx    PF1              ; 3
    stx    PF2              ; 3
    inx                     ; 2                x = 0
    stx    ENABL            ; 3                clear ball and graphics registers
    stx    GRP0             ; 3
    stx    GRP1             ; 3
    stx    GRP0             ; 3

; show animated copyright:
    stx    temp3            ; 3                show nothing before copyright
    ldy    #COPYRIGHT_H/2   ; 2
    lda    noGameScroll     ; 3
    ldx    soundIdx         ; 3
    beq    .noSound0        ; 2
    lda    #0               ; 2
.noSound0:
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2
    cmp    #20              ; 2                scroll-animation
    bcs    .ok              ; 2
    ldy    #0               ; 2
    cmp    #12              ; 2
    bcc    .ok              ; 2
    sbc    #12              ; 2
    tay                     ; 2
.ok:
    tya                     ; 2
    clc                     ; 2
    adc    #<CopyRight5-COPYRIGHT_H/2; 2
    ldx    #12-2            ; 2
.loopCopyright:
    sta    WSYNC            ; 3
    sta    HMOVE            ; 3
    sta    digitPtr,x       ; 4
    sec                     ; 2
    sbc    #COPYRIGHT_H     ; 2
    dex                     ; 2
    dex                     ; 2
    bpl    .loopCopyright   ; 2

    lda    colorLst+4       ; 3
    sta    COLUPF           ; 3
    jsr    ShowDigits       ; 6
    lda    noGameScroll     ; 3                game running?
    beq    .endCopyright    ; 2                 yes, no more scrolling
    dec    noGameScroll     ; 5                 no, scoll message
```

```
    bne    .endCopyright    ; 2
    dec    noGameScroll     ; 5                avoid #0
.endCopyright:

; start timer and vertical blank:
  IF NTSC
    lda    #32              ; 2
  ELSE
    lda    #60
  ENDIF
    ldx    #%10000010       ; 2
    sta    WSYNC            ; 3
    sta    TIM64T           ; 4
    stx    VBLANK           ; 3

; check for killed Harry:
    lda    soundIdx         ; 3
    cmp    #SOUND_FALLING-1 ; 2                dead tune at end of playing?
    bne    .slipDecrease    ; 2                 no, skip decrease
; Harry is loosing a live:
    lda    livesPat         ; 3                any more lives?
    beq    .slipDecrease    ; 2                 no, skip decrease
  IF TRAINER
    FILL_NOP 4
  ELSE
    asl                     ; 2                 yes, decrease lives
    asl                     ; 2
    sta    livesPat         ; 3
  ENDIF
    lda    #NOREFLECT       ; 2
    sta    reflectHarry     ; 3
    sta    noGameScroll     ; 3
    sta    CXCLR            ; 3
    ldy    #$d0|NO_MOVE     ; 2                upper Harry restart y-position
    sty    oldJoystick      ; 3                clear joystick
    lda    #20              ; 2
    sta    xPosHarry        ; 3
    ldx    #JUMP_LEN        ; 2
    lda    yPosHarry        ; 3
    cmp    #71              ; 2                Harry at underground?
    bcc    LF5D2            ; 2                 no, skip
    ldy    #64              ; 2                 yes, lower Harry restart y-
position
    lda    #SCREENWIDTH/2-4 ; 2                position scorpion at center..
    sta    xPosScorpion     ; 3                ..when Harry restarts at
underground
LF5D2:
    stx    jumpIndex        ; 3
    sty    yPosHarry        ; 3
.slipDecrease:

; *** sound routines: ***
    ldy    #0               ; 2
    ldx    soundIdx         ; 3
    beq    .noSound         ; 2
    inc    soundDelay       ; 5
    lda    soundDelay       ; 3
    and    #$03             ; 2                next note every 4th frame
```

```
    bne    .skipNext        ; 2
    inc    soundIdx         ; 5                 play next note
.skipNext:
    lda    SoundTab-1,x     ; 4
    bpl    .contSound       ; 2
    sty    soundIdx         ; 3                 stop current sound
.contSound:
    sta    AUDF0            ; 3
    ldy    #1               ; 2
.noSound:
    sty    AUDC0            ; 3
    lda    #4               ; 2
    sta    AUDV0            ; 3

; check if Harry has fallen into a hole or pit:
    lda    climbPos         ; 3                 Harry at ladder?
    bne    .exitBounds      ; 2+1                yes, skip bounds check
    lda    yPosHarry        ; 3
    cmp    #JUNGLE_GROUND   ; 2                 Harry at ground?
    bne    .exitBounds      ; 2                  no, skip bounds check
    ldx    sceneType        ; 3
    cpx    #CROCO_SCENE     ; 2                 croco scene?
    bne    .noCroco1        ; 2                  no, skip
    bit    frameCnt         ; 3                 open croco jaws?
    bpl    .contCroco       ; 2                  yes, skip
    dex                     ; 2                  no, use other values
    bne    .contCroco       ; 3

.noCroco1:
    cpx    #HOLE3_SCENE+2   ; 2                 scene with hole(s) or ???? ?
    bcc    .contCroco       ; 2                  yes, skip
    ldx    #HOLE3_SCENE+1   ; 2                  no, limit scene type
.contCroco:
    txa                     ; 2
    asl                     ; 2
    asl                     ; 2
    asl                     ; 2
    tax                     ; 2
    ldy    #3               ; 2                 check up to 4 bounds
.loopBounds:
    lda    HoleBoundsTab,x  ; 4
    beq    .exitBounds      ; 2                 no more bounds!
    clc                     ; 2
    adc    xPosQuickSand    ; 3
    cmp    xPosHarry        ; 3                 Harry left of hole/pit?
    bcs    .inBounds        ; 2                  yes, bound ok
    lda    HoleBoundsTab+1,x; 4
    sec                     ; 2
    sbc    xPosQuickSand    ; 3
    cmp    xPosHarry        ; 3                 Harry right of hole/pit?
    bcs    .outOfBounds     ; 2                  no, Harry is falling into
.inBounds:
    inx                     ; 2
    inx                     ; 2
    dey                     ; 2
    bpl    .loopBounds      ; 2
    bmi    .exitBounds      ; 3
```

```
.outOfBounds:
    inc    yPosHarry        ; 5                Harry is falling down
    ldx    #JUMP_LEN        ; 2
    stx    jumpIndex        ; 3
    dex                     ; 2
    stx    oldJoystick      ; 3                x=$1f -> no direction
.exitBounds:
    lda    jumpMode         ; 3                JTZ: superfluous code?
    bne    .waitTim         ; 2@
    bit    hitLiana         ; 3                collison with liana
    bvc    .waitTim         ; 2@                no, skip
    lda    jumpIndex        ; 3                currently jumping?
    beq    .waitTim         ; 2@                no, skip
    ldx    atLiana          ; 3                Harry already at liana?
    bne    .waitTim         ; 2@                yes, skip
    stx    jumpIndex        ; 3                 no, stop jump
    inx                     ; 2
    stx    atLiana          ; 3                enter "liana mode"
    stx    soundIdx         ; 3                start tarzan sound (=0)

; wait for end of vertical blank:
.waitTim:
    lda    INTIM            ; 4
    bne    .waitTim         ; 2@

; start vertical sync:
    sta    AUDC1            ; 3
    ldy    #%10000010       ; 2                enable vertical sync and dump
ports (JTZ: why?)
    sty    WSYNC            ; 3
    sty    VSYNC            ; 3
    sty    WSYNC            ; 3
    sty    WSYNC            ; 3
    sty    WSYNC            ; 3
    sta    VSYNC            ; 3

  IF SCREENSAVER
; process screensaver code:
    inc    SS_DelayLo       ; 5
    bne    .skipSS_Delay    ; 2@
    inc    SS_Delay         ; 5
    bne    .skipSS_Delay    ; 2@
    sec                     ; 2
    ror    SS_Delay         ; 5
.skipSS_Delay:
    ldy    #$ff             ; 2
    lda    SWCHB            ; 4
    and    #BW_MASK         ; 2
    bne    .colorMode       ; 2@
    ldy    #$0f             ; 2
.colorMode:
    tya                     ; 2
    ldy    #$00             ; 2                disable changing colors
    bit    SS_Delay         ; 3
    bpl    .noScreenSaver   ; 2@
    and    #$f7             ; 2                avoid bright colors in screensaver
mode
    ldy    SS_Delay         ; 3
```

```
.noScreenSaver:
    sty    SS_XOR           ; 3
    asl    SS_XOR           ; 5
    sta    SS_Mask          ; 3
  ELSE
    FILL_NOP 39
  ENDIF

; start timer for vertical sync:
  IF NTSC
    lda    #47              ; 2
  ELSE
    lda    #79
  ENDIF
    sta    WSYNC            ; 3
    sta    TIM64T           ; 4

; read joystick:
    lda    SWCHA            ; 4
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2
    sta    joystick         ; 3
    cmp    #NO_MOVE         ; 2
    beq    .noMove          ; 2
    ldx    #0               ; 2
  IF SCREENSAVER
    stx    SS_Delay         ; 3                reset screensaver
  ELSE
    FILL_NOP 2
  ENDIF
    lda    timerHi          ; 3
    cmp    #STARTTIME       ; 2                timer at 20:00?
    bne    .noMove          ; 2                 no, skip
    stx    noGameScroll     ; 3                 yes, game is running
.noMove:

; read RESET switch:
    lda    SWCHB            ; 4
    lsr                     ; 2                RESET pressed?
    bcs    .noReset         ; 2                 no, skip
  IF SCREENSAVER
    ldx    #SS_Delay        ; 2                 yes, load init-values offset..
  ELSE
    ldx    #frameCnt        ; 2                 yes, load init-values offset..
  ENDIF
    jmp    Reset            ; 3                ..and jump to Reset

.noReset:
    lda    noGameScroll     ; 3                game running?
    beq    .processHarry    ; 2                 yes, process Harry
    jmp    ProcessObjects   ; 3                 no, skip Harry, goto objects

; *** process Harry: ***
.processHarry:
    inc    frameCnt         ; 5
    lda    random2          ; 3
```

```
    asl                     ; 2
    eor    random2          ; 3
    asl                     ; 2
    rol    random2          ; 5
    lda    climbPos         ; 3                Harry at ladder?
    bne    .endDoJump       ; 2*+1              no, skip continue jump
    ldx    jumpIndex        ; 3                currently jumping?
    beq    .endDoJump       ; 2*+1              no, skip continue jump
    lda    yPosHarry        ; 3                 yes, calculate..
    sec                     ; 2                ..new y-position of Harry
    sbc    JumpTab-1,x      ; 4
    sta    yPosHarry        ; 3
    inc    jumpIndex        ; 5
    lda    jumpIndex        ; 3
    cmp    #JUMP_LEN+1      ; 2
    bcc    .indexOk         ; 2*
    lda    #JUMP_LEN        ; 2
    sta    jumpIndex        ; 3
.indexOk:

    ldx    yPosHarry        ; 3
    cpx    #JUNGLE_GROUND   ; 2                Harry at jungle ground?
    beq    .stopJump        ; 2*+1              yes, stop any jump
    ldy    ladderFlag       ; 3                ladder in scene?
    beq    .skipFalling     ; 2*+1              no, skip falling
    cpx    #JUNGLE_GROUND+2 ; 2
    bne    .skipFalling     ; 2*+1
    lda    #SOUND_FALLING   ; 2                Harry is falling into a hole
    sta    soundIdx         ; 3                start falling-sound
    lda    #$00             ; 2
    jsr    DecScoreHi       ; 6                subtract 100 points from score
.skipFalling:
    cpx    #UNDER_GROUND    ; 2                is Harry at underground bottom?
    beq    .stopJump        ; 2*               yes, stop any jump
    cpx    #54              ; 2                has Harry reached the falling
limit?
    bne    .endDoJump       ; 2*               no, skip
    tya                     ; 2                ladder in scene?
    bne    .endDoJump       ; 2*               no, skip kill
    jmp    KilledHarry      ; 3                yes, Harry is killed

.stopJump:
    lda    #0               ; 2
    sta    jumpIndex        ; 3
    sta    jumpMode         ; 3
.endDoJump:

; countdown timer:
    dec    timerLo          ; 5
    bpl    .inTime          ; 2*
    lda    #FRAMERATE-1     ; 2
    sta    timerLo          ; 3
    sed                     ; 2
    lda    timerMed         ; 3
    sec                     ; 2
    sbc    #$01             ; 2
    bcs    .contMinute      ; 2*
    lda    #$59             ; 2                start next minute
```

```
.contMinute:
    sta    timerMed         ; 3
    lda    timerHi          ; 3
  IF TRAINER
    lda    #$19             ; 2
  ELSE
    sbc    #$00             ; 2
  ENDIF
    sta    timerHi          ; 3
    cld                     ; 2
    lda    timerHi          ; 3                 any more..
    ora    timerMed         ; 3                 ..time left?
    bne    .inTime          ; 2                  yes, continue
    dec    noGameScroll     ; 5                  no, stop game
.inTime:

; check collisions between Harry and object:
    lda    CXPPMM-$30       ; 3                 Harry collided?
    bmi    .contCollision   ; 2                  yes, process collisions
    lda    #0               ; 2                  no, skip collisions
    sta    patOfsHarry      ; 3
    beq    .endCollision    ; 3

.contCollision:
    lda    yPosHarry        ; 3
    cmp    #64              ; 2                 Harry at underground?
    bcs    .checkWallHit    ; 2                  yes, check wall
    lda    atLiana          ; 3                  no, Harry at liana?
    bne    .endCollision    ; 2                  yes, skip
    lda    sceneType        ; 3
    cmp    #CROCO_SCENE     ; 2                 croco in scene?
    beq    .endCollision    ; 2                  yes, skip
    cmp    #TREASURE_SCENE  ; 2                 treasue in scene?
    bne    .noTreasure1     ; 2                  no, skip
    jsr    CheckTreasures   ; 6                  yes, check if treasure was found
before
    bne    .endCollision    ; 2
    sta    treasureBits,x   ; 4                 clear treasure bit
    dec    treasureCnt      ; 5                 all treasures found
    bpl    .incScore        ; 2                  no, skip
    dec    noGameScroll     ; 5                  yes, game finished!!!

; treasure found, increase score:
.incScore:
    lda    objectType       ; 3
    and    #$03             ; 2
    asl                     ; 2
    asl                     ; 2
    asl                     ; 2
    asl                     ; 2
    adc    #$20             ; 2                 add at least 2000 points
    sed                     ; 2
    adc    scoreMed         ; 3
    sta    scoreMed         ; 3
    lda    #$00             ; 2
    adc    scoreHi          ; 3
    sta    scoreHi          ; 3
    cld                     ; 2
```

```
    lda    #SOUND_TREASURE  ; 2
    sta    soundIdx         ; 3
    bne    .endCollision    ; 3

.noTreasure1:
    lda    objectType       ; 3
    cmp    #ID_FIRE         ; 2                fire or cobra?
    bcc    .hitLogs         ; 2                 no, hit by rolling logs
.noWallHit:
    jmp    KilledHarry      ; 3                Harry is killed

.hitLogs:
    lda    climbPos         ; 3                Harry at ladder?
    beq    .notAtLadder     ; 2                 no, skip push
    inc    climbPos         ; 5                 yes, push down Harry
    bne    .decScore        ; 3

.notAtLadder:
    lda    yPosHarry        ; 3
    cmp    #JUNGLE_GROUND+1 ; 2
    bcs    .endCollision    ; 2
    lda    #5               ; 2
    sta    patOfsHarry      ; 3
    lda    objectType       ; 3
    and    #$04             ; 2
    bne    .decScore        ; 2
    lda    #NO_MOVE         ; 2
    sta    joystick         ; 3
.decScore:
    jsr    DecScoreLo       ; 6
.endCollision:
    jmp    .swingLiana      ; 3

.checkWallHit:
    lda    wallPatPtr       ; 3
    cmp    #<Wall           ; 2                wall displayed in scene?
    bne    .noWallHit       ; 2                 no, skip
    lda    #$01             ; 2                 yes, make some noise
    sta    AUDC1            ; 3
    lda    xPosHarry        ; 3                determine where Harry hit the wall
    cmp    #140             ; 2                right wall from the right?
    bcs    .hitFromRight    ; 2                 yes, continue
    cmp    #13              ; 2                left wall from the left?
    bcc    .hitFromLeft     ; 2                 yes, continue
    cmp    #80              ; 2                left or right wall?
    bcs    .hitFromLeft     ; 2
.hitFromRight:
    inc    xPosHarry        ; 5                bounce back one pixel and..
    ldx    #MOVE_RIGHT      ; 2                ..change direction to right
    bne    .contWallHit     ; 3

.hitFromLeft:
    dec    xPosHarry        ; 5                bounce back one pixel and..
    ldx    #MOVE_LEFT       ; 2                ..change direction to left
.contWallHit:
    stx    oldJoystick      ; 3

; let the liana swing:
```

```
.swingLiana:
; calculate absolute position:
    lda    lianaPosLo       ; 3
    asl                     ; 2
    lda    lianaPosHi       ; 3
    rol                     ; 2
    bpl    .skipNeg         ; 2
    eor    #$ff             ; 2
.skipNeg:
    sta    hmblAdd          ; 3                store absolute value (-> angle of
liana)
    ldy    #$f0             ; 2                liana moves right
  IF OPTIMIZE
    bcs    .skipMoveLeft    ; 2
    FILL_NOP 2
  ELSE
    lda    lianaPosHi       ; 3
    bmi    .skipMoveLeft    ; 2
  ENDIF
    ldy    #$10             ; 2                liana moves left
.skipMoveLeft:
    sty    hmblDir          ; 3
    sec                     ; 2
    lda    #143             ; 2
    sbc    hmblAdd          ; 3
    clc                     ; 2
    adc    lianaPosLo       ; 3
    sta    lianaPosLo       ; 3
    bcc    .skipAddHi       ; 2
    lda    lianaPosHi       ; 3
    adc    #3               ; 2
    sta    lianaPosHi       ; 3
.skipAddHi:
; calculate bottom of liana:
    lda    hmblAdd          ; 3                this are no exactly maths,..
    lsr                     ; 2                ..but who cares, as long..
    lsr                     ; 2                ..as it's looking ok :)
    lsr                     ; 2
    cmp    #6-1             ; 2
    bcs    .limitBottom     ; 2                limit bottom of liana to 6
    lda    #6               ; 2
.limitBottom:
    adc    #4               ; 2
    sta    lianaBottom      ; 3

; check for a new jump:
    lda    jumpIndex        ; 3                currently jumping?
    beq    .notJumping      ; 2                 no,
    cmp    #3               ; 2                jump just started?
    bcc    .saveDir         ; 2                 yes, save joystick direction
.notJumping:
    ora    climbPos         ; 3                Harry at ladder..
    ora    patOfsHarry      ; 3                ..or Harry kneeing..
    ora    atLiana          ; 3                ..or Harry at liana?
    bne    .noFire          ; 2                 yes, skip new jump
    lda    INPT4-$30        ; 3
    and    #%10000000       ; 2
    cmp    fireButton       ; 3
```

```
    sta    fireButton       ; 3
    beq    .noFire          ; 2
    tax                     ; 2
    bmi    .noFire          ; 2
; start jump:
    lda    #1               ; 2                start jumping sequence
    sta    jumpIndex        ; 3
  IF SCREENSAVER
    sta    SS_Delay         ; 3
  ELSE
    FILL_NOP 2
  ENDIF
    lda    #SOUND_JUMP      ; 2
    sta    soundIdx         ; 3
    dec    yPosHarry        ; 5                move Harry up
.saveDir:
    lda    joystick         ; 3
    sta    oldJoystick      ; 3
.noFire:

; check for jumping of liana:
    lda    atLiana          ; 3                Harry at liana?
    beq    .skipJumpOff     ; 2                 no, skip jump of liana
    lda    joystick         ; 3
    and    #~[$f0|MOVE_DOWN]; 2                joystick down?
    bne    .skipJumpOff     ; 2                 no, skip
    sta    atLiana          ; 3                 yes, leave "liana mode"
    lda    #JUMP_LEN/2      ; 2                start jump down
    sta    jumpIndex        ; 3
    sta    jumpMode         ; 3
    ldy    #MOVE_RIGHT      ; 2
    lda    hmblDir          ; 3                jump in liana direction
    bmi    .jumpRight       ; 2
    ldy    #MOVE_LEFT       ; 2
.jumpRight:
    sty    oldJoystick      ; 3
.skipJumpOff:

; check for starting climbing ladder:
    lda    climbPos         ; 3                Harry at ladder?
    bne    .endStartClimb   ; 2                 yes, skip
    lda    ladderFlag       ; 3                ladder in scene?
    beq    .endStartClimb   ; 2                 no, skip
    lda    xPosHarry        ; 3
    sec                     ; 2
    sbc    #68              ; 2
    cmp    #15              ; 2                Harry at x-position of ladder (+/-
7)?
    bcs    .endStartClimb   ; 2                 no, skip
    lda    yPosHarry        ; 3                 yes,
    cmp    #84              ; 2                Harry near bottom of underground
    bcc    .skipClimbUp     ; 2                 no, skip
    lda    joystick         ; 3                 yes,
    lsr                     ; 2                joystick up?
    bcs    .skipClimbUp     ; 2                 no, skip
    lda    #LADDER_BOTTOM-1 ; 2                 yes, start climbing up the ladder
    bne    .contClimbUp     ; 3
```

```
.skipClimbUp:
    lda    yPosHarry        ; 3
    cmp    #JUNGLE_GROUND   ; 2               Harry at jungle-ground?
    bne    .endStartClimb   ; 2               no, skip
    lda    joystick         ; 3
    and    #~[$f0|MOVE_DOWN]; 2               joystick down?
    bne    .endStartClimb   ; 2               no, skip
    lda    #LADDER_TOP+1    ; 2               yes, start climbing down the
ladder
.contClimbUp:
    sta    climbPos         ; 3
    lda    #SCREENWIDTH/2-4 ; 2               set x-position of..
    sta    xPosHarry        ; 3               ..Harry at ladder
.endStartClimb:

; move Harry when swinging at liana:
    lda    atLiana          ; 3               Harry at liana?
    beq    .skipSwingHarry  ; 2               no, skip moving Harry
; set x-position of Harry:
    lda    hmblAdd          ; 3
    lsr                     ; 2
    lsr                     ; 2
    clc                     ; 2
    ldy    hmblDir          ; 3
    bmi    .isNeg           ; 2
    eor    #$ff             ; 2               negate
    sec                     ; 2
.isNeg:
    adc    #75              ; 2
    sta    xPosHarry        ; 3
; set y-position of Harry:
    lda    #$29             ; 2
    sec                     ; 2
    sbc    lianaBottom      ; 3
    sta    yPosHarry        ; 3
.skipSwingHarry:

; check Harry climbing ladder:
    lda    climbPos         ; 3               Harry at ladder?
    beq    .endClimbLadder  ; 2               no, skip
    lda    #0               ; 2
    sta    jumpIndex        ; 3
    lda    frameCnt         ; 3
    and    #$07             ; 2               climb every 8th frame
    bne    .skipAnimClimb   ; 2
    lda    joystick         ; 3
    lsr                     ; 2               joystick up?
    bcs    .notClimbUp      ; 2               no, skip
    dec    climbPos         ; 5               yes, climb up
.notClimbUp:
    lsr                     ; 2               joystick down?
    bcs    .notClimbDown    ; 2               no, skip
    inc    climbPos         ; 5               yes, climb down
.notClimbDown:
    lda    climbPos         ; 3
    cmp    #LADDER_TOP      ; 2               top reached?
    bcs    .skipLadderTop   ; 2               no, skip
    lda    #NO_MOVE         ; 2
```

```
    sta    oldJoystick      ; 3
    lda    #LADDER_TOP      ; 2
.skipLadderTop:
    cmp    #LADDER_BOTTOM   ; 2               bottom reached?
    bcc    .skipLadderBottom; 2/3              no, skip
    lda    #0               ; 2               remove Harry from ladder
    ldx    #ID_STANDING     ; 2
    stx    patIdHarry       ; 3
    ldx    #SCREENWIDTH/2+6 ; 2
    stx    yPosHarry        ; 3
.skipLadderBottom:
    sta    climbPos         ; 3
.skipAnimClimb:
    lda    climbPos         ; 3               Harry at ladder?
    beq    .endClimbLadder  ; 2/3              no, skip
    asl                     ; 2               yes, calculate y-position of
Harry
    sec                     ; 2
    rol                     ; 2
    adc    #1               ; 2
    sta    yPosHarry        ; 3
.endClimbLadder:

; animate running Harry:
    lda    atLiana          ; 3               Harry at liana?
    bne    .endHarryId      ; 2/3+1            yes, skip running
    lda    climbPos         ; 3
    cmp    #LADDER_TOP+1    ; 2               Harry at ladder bottom?
    bcs    .endHarryId      ; 2/3+1            no, skip running
    lda    frameCnt         ; 3               animate Harry (every 4th frame)
    and    #$03             ; 2
    tax                     ; 2
    lsr                     ; 2
    bcs    .endHarryId      ; 2/3+1
    lda    oldJoystick      ; 3
    ldy    jumpIndex        ; 3               currently jumping?
    bne    .isJumping       ; 2/3              yes, use old joystick input
    lda    joystick         ; 3                no, use new joystick input
.isJumping:
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2               joystick left?
    bcs    .skipLeft        ; 2/3+1            no, skip
    dec    xPosHarry        ; 5
    ldy    #REFLECT         ; 2
    sty    reflectHarry     ; 3
    cpx    #0               ; 2               4th frame?
    bne    .skipAnimLeft    ; 2/3              no, skip animation
    dec    patIdHarry       ; 5
.skipAnimLeft:
    jmp    .endAnimHarry    ; 3

.skipLeft:
    lsr                     ; 2               joystick right?
    bcs    .endAnimHarry    ; 2/3              no, skip
    inc    xPosHarry        ; 5
    ldy    #NOREFLECT       ; 2
    sty    reflectHarry     ; 3
```

```
    cpx    #0               ; 2                 4th frame?
    bne    .endAnimHarry    ; 2*                 no, skip animation
    dec    patIdHarry       ; 5
.endAnimHarry:

; goto next scene, if Harry has reached border of current scene:
    ldx    #0               ; 2                 move one scene
    lda    yPosHarry        ; 3
    cmp    #64              ; 2                 Harry at underground?
    bcc    .oneScene        ; 2*                 yes, move one scene
    ldx    #2               ; 2                  no, move three scenes
.oneScene:
    lda    xPosHarry        ; 3
    cmp    #XMIN_HARRY      ; 2
    bcs    .notAtLeft       ; 2*
    jsr    LeftRandom       ; 6
    lda    #XMAX_HARRY      ; 2
    sta    xPosHarry        ; 3
.notAtLeft:
    cmp    #XMAX_HARRY+1    ; 2
    bcc    .notAtRight      ; 2*
    jsr    RightRandom      ; 6
    lda    #XMIN_HARRY      ; 2
    sta    xPosHarry        ; 3
.notAtRight:
    lda    patIdHarry       ; 3                 illegal animation id?
    bpl    .endHarryId      ; 2*                 no, skip
    lda    #ID_RUNNING4     ; 2                  yes, start new animation sequence
    sta    patIdHarry       ; 3
.endHarryId:

; move the scorpion towards harry:
    lda    ladderFlag       ; 3                 ladder in scene?
    bne    .noMoveScorpion  ; 2*                 yes, skip scorpion
    ldx    #NOREFLECT       ; 2
    lda    xPosHarry        ; 3
    sec                     ; 2
    sbc    xPosScorpion     ; 3
    beq    .noMoveScorpion  ; 2*
    bcs    .rightOfScorpion ; 2*
    ldx    #REFLECT         ; 2
.rightOfScorpion:
    lda    frameCnt         ; 3
    and    #$07             ; 2                 move scorpion every 8th frame
    bne    .endMoveScorpion ; 2*
    inc    xPosScorpion     ; 5
    bcs    .endMoveScorpion ; 2*
    dec    xPosScorpion     ; 5
    dec    xPosScorpion     ; 5
.endMoveScorpion:
    stx    reflectScorpion  ; 3
.noMoveScorpion:

    lda    climbPos         ; 3
    cmp    #LADDER_TOP      ; 2                 Harry at ladder top?
    bne    .notAtTop        ; 2*                 no, skip horizontal move
    lda    joystick         ; 3
    and    #JOY_HORZ        ; 2
```

```
    cmp    #JOY_HORZ        ; 2                joystick left or right?
    beq    .notAtTop        ; 2�                no, skip
    lda    joystick         ; 3
    sta    oldJoystick      ; 3
    lda    #1               ; 2
    sta    jumpIndex        ; 3
    lsr                     ; 2
    sta    climbPos         ; 3                remove Harry from ladder (=0)
    lda    #31              ; 2                ..and remark this state
    sta    yPosHarry        ; 3
.notAtTop:
    ldx    patIdHarry       ; 3
    lda    joystick         ; 3
    and    #JOY_HORZ        ; 2
    cmp    #JOY_HORZ        ; 2                joystick left or right??
    bne    .skipStanding    ; 2�                yes, skip
    ldx    #ID_STANDING     ; 2                no, draw standing Harry
.skipStanding:
    lda    yPosHarry        ; 3
    cmp    #31              ; 2               Harry just jumping of top of
ladder?
    bne    .notOfTop        ; 2�                no, skip
    ldx    #3               ; 2                yes, draw diffenret shape
    bne    .contPatId       ; 3

.notOfTop:
    cmp    #UNDER_GROUND    ; 2
    beq    .contPatId       ; 2�
    cmp    #JUNGLE_GROUND   ; 2
    beq    .contPatId       ; 2�
    ldx    #ID_KNEEING      ; 2
    bcc    .contPatId       ; 2�
    cmp    #60              ; 2
    bcs    .contPatId       ; 2�
    lda    climbPos         ; 3               Harry at ladder?
    bne    .contPatId       ; 2�                yes, skip
    ldx    #ID_STANDING     ; 2                no, draw standing Harry
.contPatId:
    lda    atLiana          ; 3               Harry at liana?
    beq    .skipStanding2   ; 2�                no, skip
    ldx    #ID_SWINGING     ; 2                yes, draw swinging Harry
.skipStanding2:
    lda    climbPos         ; 3               Harry at ladder?
    beq    .noAnimClimb     ; 2�                no, skip
    and    #%1              ; 2                yes, animate climbing
    clc                     ; 2
    adc    #ID_CLIMBING     ; 2
    tax                     ; 2
.noAnimClimb:
    lda    patOfsHarry      ; 3               Harry hit by logs (kneeing)?
    beq    .skipKneeing     ; 2�                no, skip
    ldx    #ID_KNEEING      ; 2                yes, draw kneeing Harry
.skipKneeing:
    stx    patIdHarry       ; 3
    lda    HarryPtrTab,x    ; 4
    sta    harryPatPtr      ; 3
    lda    #>Harry0         ; 2
    sta    harryPatPtr+1    ; 3
```

```
    lda    #<RunColTab      ; 2
    cpx    #ID_SWINGING+1   ; 2
    bcc    .runColors       ; 2
    lda    #<ClimbColTab    ; 2
.runColors:
    sta    harryColPtr      ; 3

ProcessObjects SUBROUTINE
    lda    objectType       ; 3
    tax                     ; 2
    ldy    sceneType        ; 3
    lda    LianaTab,y       ; 4
    sta    ENABL            ; 3
    cpy    #TREASURE_SCENE  ; 2                treasure in scene?
    bne    .noTreasure      ; 2                 yes, skip
    jsr    CheckTreasures   ; 6
    beq    .withTreasure    ; 2
    ldx    #ID_NOTHING      ; 2
    bne    .noTreasure      ; 3

.withTreasure:
    lda    objectType       ; 3
    and    #$03             ; 2
    ora    #$08             ; 2                treasures only
    tax                     ; 2
.noTreasure:

; animate some treasures and hazards:
    lda    random2          ; 3
    and    AnimateTab,x     ; 4
    sta    temp1            ; 3
    lda    ObjectPtrTab,x   ; 4
    clc                     ; 2
    adc    temp1            ; 3
    sta    objPatPtr        ; 3

    lda    NuSize1Tab,x     ; 4
    sta    nusize1          ; 3                save number of object copies
    lda    Color1PtrTab,x   ; 4
    sta    objColPtr        ; 3
; special processing for crocodiles:
    ldy    sceneType        ; 3
    lda    CrocoTab,y       ; 4
    beq    .noCroco         ; 2
    lda    #<Croco0         ; 2
    bit    frameCnt         ; 3                open croco jaws?
    bpl    .skipClosed      ; 2                 yes, skip
    lda    #<Croco1         ; 2                 no, use other shape
.skipClosed:
    sta    objPatPtr        ; 3
    lda    #<CrocoColor     ; 2
    sta    objColPtr        ; 3
    lda    objectType       ; 3
    sta    ENABL            ; 3
    lda    #THREE_COPIES    ; 2
    sta    nusize1          ; 3
.noCroco:
    lda    noGameScroll     ; 3                game running?
```

```
    bne    .skipLogs        ; 2¹                no, skip
    lda    CrocoTab,y       ; 4                crocos in scene?
    bne    .skipLogs        ; 2¹                no, skip
    cpy    #TREASURE_SCENE  ; 2                treasure in scene?
    beq    .skipLogs        ; 2¹                yes, skip

; animate, bounce and move rolling logs:
    lda    xPosObject       ; 3
    asl                     ; 2
    asl                     ; 2
    asl                     ; 2
    and    #$30             ; 2                rolling logs bounce for..
    cmp    #$30             ; 2                ..2 pixel every 8th x-position
    and    #$10             ; 2
    adc    objPatPtr        ; 3
    sta    objPatPtr        ; 3
    lda    frameCnt         ; 3
    lsr                     ; 2                move logs every 2nd frame
    bcs    .skipLogs        ; 2¹
    lda    objectType       ; 3
    cmp    #ID_STATIONARY   ; 2                rolling logs scene?
    bcs    .skipLogs        ; 2¹                no, skip move
    ldx    xPosObject       ; 3
    bne    .skipResetLogs   ; 2¹
    ldx    #SCREENWIDTH     ; 2
.skipResetLogs:
    dex                     ; 2
    stx    xPosObject       ; 3
.skipLogs:

; set score pointers, replace leading zeros with space:
    jsr    SetDigitPtrs     ; 6
    inx                     ; 2                x = 0
.loopSpace:
    lda    digitPtr,x       ; 4
    bne    .exitSpace       ; 2¹
    lda    #<Space          ; 2
    sta    digitPtr,x       ; 4
    inx                     ; 2
    inx                     ; 2
    cpx    #9               ; 2
    bcc    .loopSpace       ; 2¹
.exitSpace:
    jmp    MainLoop         ; 3


InitGame SUBROUTINE
    ldx    #1               ; 2
    stx    random2          ; 3
.loopInitSound:
    lda    #$04             ; 2                init both sound channels
    sta    AUDV0,x          ; 4
    lda    #$10             ; 2
    sta    AUDF0,x          ; 4
    dex                     ; 2
    bpl    .loopInitSound   ; 2¹

    stx    noGameScroll     ; 3                game is stopped
```

```
    sta    xPosHarry        ; 3
    asl                     ; 2
    sta    yPosHarry        ; 3
    sta    scoreMed         ; 3                 = $20
  IF STARTTIME != $20
    lda    #STARTTIME
  ENDIF
    sta    timerHi          ; 3

    ldx    #27              ; 2
.loopInit:
    lda    InitTab,x        ; 4
    sta    harryPatPtr,x    ; 4
    dex                     ; 2
    bpl    .loopInit        ; 2

    lda    #FRAMERATE-1     ; 2
    sta    timerLo          ; 3
    lda    #31              ; 2
    sta    treasureCnt      ; 3
    lda    #%10100000       ; 2                 3 lives
    sta    livesPat         ; 3
    lda    #RAND_SEED       ; 2                 set starting scene
    sta    random           ; 3
    bne    ContRandom       ; 3

LeftRandom SUBROUTINE
; generate new random scene on the left:
.loopRandom:
; random' = random >> 1 | (bit4^bit5^bit6^bit1) * $80
    lda    random           ; 3
    asl                     ; 2
    eor    random           ; 3
    asl                     ; 2
    eor    random           ; 3
    asl                     ; 2
    asl                     ; 2
    rol                     ; 2
    eor    random           ; 3
    lsr                     ; 2
    ror    random           ; 5
    dex                     ; 2
    bpl    .loopRandom      ; 2
ContRandom:
    lda    #124             ; 2                 x-position of logs, fire, cobra or
treasure
    sta    xPosObject       ; 3
    lda    random           ; 3
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2
    pha                     ; 3
    and    #%111            ; 2
    sta    sceneType        ; 3                 bits 3..5
    pla                     ; 4
    lsr                     ; 2
    lsr                     ; 2
    lsr                     ; 2
```

```
    sta    treePat          ; 3                bits 6 & 7
    lda    random           ; 3
    and    #%111            ; 2
    sta    objectType       ; 3                bits 0..2
    ldx    #SCREENWIDTH/2-4 ; 2                center x-position of scorpion
    ldy    #NOLADDER        ; 2
    lda    sceneType        ; 3
    cmp    #HOLE3_SCENE+1   ; 2                scene with hole(s)?
    bcs    .setFlag         ; 2                 no, skip
    ldy    #WITHLADDER      ; 2                 yes, enable ladder
    ldx    #17              ; 2                left wall x-position
    lda    random           ; 3
    asl                     ; 2                position of the wall? (bit 7)
    bcc    .setFlag         ; 2                 left, skip
    ldx    #136             ; 2                 right wall x-position
.setFlag:
    sty    ladderFlag       ; 3
    stx    xPosScorpion     ; 3                also used for wall position

    ldx    sceneType        ; 3
    lda    CrocoTab,x       ; 4
    beq    .noCrocos        ; 2
    lda    #60              ; 2                x-position crocos
    sta    xPosObject       ; 3
.noCrocos:
    rts                     ; 6


    align 256

Harry0:
    .byte %00000000 ; |        |
    .byte %00000000 ; |        |
    .byte %00000000 ; |        |
    .byte %00000000 ; |        |
    .byte %00000000 ; |        |
    .byte %00110011 ; |  XX  XX|
    .byte %01110010 ; | XXX  X |
    .byte %11011010 ; |XX XX X |
    .byte %00011110 ; |   XXXX |
    .byte %00011100 ; |   XXX  |
    .byte %00011000 ; |   XX   |
    .byte %01011000 ; | X XX   |
    .byte %01011000 ; | X XX   |
    .byte %01111100 ; | XXXXX  |
    .byte %00111110 ; |  XXXXX |
    .byte %00011010 ; |   XX X |
    .byte %00011000 ; |   XX   |
    .byte %00010000 ; |   X    |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00000000 ; |        |
Harry1:
    .byte %00000000 ; |        |
    .byte %10000000 ; |X       |
    .byte %10000000 ; |X       |
    .byte %11000011 ; |XX    XX|
```

```
    .byte %01100010 ; | XX   X |
    .byte %01100010 ; | XX   X |
    .byte %00110110 ; |  XX XX |
    .byte %00111110 ; |  XXXXX |
    .byte %00011100 ; |   XXX  |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00111100 ; |  XXXX  |
    .byte %00111110 ; |  XXXXX |
    .byte %00111010 ; |  XXX X |
    .byte %00111000 ; |  XXX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00010000 ; |   X    |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00000000 ; |        |
Harry2:
    .byte %00010000 ; |   X    |
    .byte %00100000 ; |  X     |
    .byte %00100010 ; |  X   X |
    .byte %00100100 ; |  X  X  |
    .byte %00110100 ; |  XX X  |
    .byte %00110010 ; |  XX  X |
    .byte %00010110 ; |   X XX |
    .byte %00011110 ; |   XXXX |
    .byte %00011100 ; |   XXX  |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011100 ; |   XXX  |
    .byte %00011100 ; |   XXX  |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00010000 ; |   X    |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00000000 ; |        |
Harry3:
    .byte %00001100 ; |    XX  |
    .byte %00001000 ; |    X   |
    .byte %00101000 ; |  X X   |
    .byte %00101000 ; |  X X   |
    .byte %00111110 ; |  XXXXX |
    .byte %00001010 ; |    X X |
    .byte %00001110 ; |    XXX |
    .byte %00011100 ; |   XXX  |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011100 ; |   XXX  |
    .byte %00011100 ; |   XXX  |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
```

```
    .byte %00011000 ; |   XX   |
    .byte %00010000 ; |   X    |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00000000 ; |        |
Harry4:
    .byte %00000000 ; |        |
    .byte %00000010 ; |      X |
    .byte %01000011 ; | X    XX|
    .byte %01000100 ; | X   X  |
    .byte %01110100 ; | XXX X  |
    .byte %00010100 ; |   X X  |
    .byte %00011100 ; |   XXX  |
    .byte %00011100 ; |   XXX  |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00111100 ; |  XXXX  |
    .byte %00111110 ; |  XXXXX |
    .byte %00111010 ; |  XXX X |
    .byte %00111000 ; |  XXX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00010000 ; |   X    |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00000000 ; |        |
Harry5:
    .byte %00011000 ; |   XX   |
    .byte %00010000 ; |   X    |
    .byte %00011100 ; |   XXX  |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011100 ; |   XXX  |
    .byte %00011110 ; |   XXXX |
    .byte %00011010 ; |   XX X |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00010000 ; |   X    |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00000000 ; |        |
Harry6:
    .byte %00000000 ; |        |
    .byte %00000000 ; |        |
    .byte %00000000 ; |        |
    .byte %00000000 ; |        |
    .byte %00000000 ; |        |
```

```
    .byte %00000000 ; |        |
    .byte %01100011 ; | XX   XX|
    .byte %11110010 ; |XXXX  X |
    .byte %11110110 ; |XXXX XX |
    .byte %11011100 ; |XX XXX  |
    .byte %11000000 ; |XX      |
    .byte %11000000 ; |XX      |
    .byte %11000000 ; |XX      |
    .byte %11000000 ; |XX      |
    .byte %11000000 ; |XX      |
    .byte %11110000 ; |XXXX    |
    .byte %11010000 ; |XX X    |
    .byte %10010000 ; |X  X    |
    .byte %11010000 ; |XX X    |
    .byte %11010000 ; |XX X    |
    .byte %11000000 ; |XX      |
    .byte %00000000 ; |        |
Harry7:
    .byte %00110000 ; |  XX    |
    .byte %00010000 ; |   X    |
    .byte %00010000 ; |   X    |
    .byte %00010000 ; |   X    |
    .byte %00010110 ; |   X XX |
    .byte %00010100 ; |   X X  |
    .byte %00010100 ; |   X X  |
    .byte %00010110 ; |   X XX |
    .byte %00010010 ; |   X  X |
    .byte %00010110 ; |   X XX |
    .byte %00011110 ; |   XXXX |
    .byte %00011100 ; |   XXX  |
    .byte %00011000 ; |   XX   |
    .byte %00111000 ; |  XXX   |
    .byte %00111000 ; |  XXX   |
    .byte %00111100 ; |  XXXX  |
    .byte %00011110 ; |   XXXX |
    .byte %00011010 ; |   XX X |
    .byte %00000010 ; |      X |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |
Harry8:
    .byte %00001100 ; |    XX  |
    .byte %00001000 ; |    X   |
    .byte %00001000 ; |    X   |
    .byte %00001000 ; |    X   |
    .byte %01101000 ; | XX X   |
    .byte %00101000 ; |  X X   |
    .byte %00101000 ; |  X X   |
    .byte %01101000 ; | XX X   |
    .byte %01001000 ; | X  X   |
    .byte %01101000 ; | XX X   |
    .byte %01111000 ; | XXXX   |
    .byte %00111000 ; |  XXX   |
    .byte %00011000 ; |   XX   |
    .byte %00011100 ; |   XXX  |
    .byte %00011100 ; |   XXX  |
    .byte %00111100 ; |  XXXX  |
    .byte %01111000 ; | XXXX   |
```

```
    .byte %01011000 ; | X XX   |
    .byte %01000000 ; | X      |
    .byte %00011000 ; |   XX   |
    .byte %00011000 ; |   XX   |

BranchTab:
    .byte %00011000 ; |   XX   |
    .byte %01111110 ; | XXXXXX |
    .byte %11011011 ; |XX XX XX|
    .byte %10011001 ; |X  XX  X|
    .byte %10011001 ; |X  XX  X|
    .byte %10011001 ; |X  XX  X|
    .byte %10011001 ; |X  XX  X|
    .byte %10011001 ; |X  XX  X|
    .byte %10011001 ; |X  XX  X|
PF1LogTab:
    .byte %10000000 ; |X       |
    .byte %00100000 ; |  X     |
    .byte %00001000 ; |    X   |
    .byte %00000100 ; |     X  |
PF2LogTab:
    .byte %00000001 ; |       X|
    .byte %00000100 ; |     X  |
    .byte %00010000 ; |   X    |
    .byte %00100000 ; |  X     |

; values for positioning branches (fine and coarse):
BranchPosLTab:
    .byte $43, $c3, $34, $f4
BranchPosRTab:
    .byte $f2, $72, $00, $40

; pattern for different the jungle grounds:
PF2PatTab:
OneHole:
    .byte %01111111 ; | XXXXXXX|        one hole
    .byte %01111111 ; | XXXXXXX|
    .byte %01111111 ; | XXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
ThreeHoles:
    .byte %01111000 ; | XXXX   |        three holes
    .byte %01111000 ; | XXXX   |
    .byte %01111000 ; | XXXX   |
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
Pit:
    .byte %00000000 ; |        |        pits
    .byte %00000001 ; |       X|
    .byte %00000011 ; |      XX|
    .byte %00001111 ; |    XXXX|
    .byte %01111111 ; | XXXXXXX|
```

```
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|

    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|

    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|
    .byte %11111111 ; |XXXXXXXX|

QuickSandSize:
    .byte 0, 4, 8, 16, 28

; Harry's colors while climbing:
ClimbColTab:
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte PINK
    .byte PINK
    .byte PINK
    .byte BROWN
; Harry's colors while running:
RunColTab:
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
```

```
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte DARK_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte YELLOW_GREEN
    .byte PINK
    .byte PINK
    .byte PINK
    .byte BROWN

SoundTab:
    .byte $13, $13, $13, $13, $13, $13, $13, $09, $0b, $0b, $0b, $0b, $0b, $0b,
$0b, $0b
    .byte   $0b, $0b, $0b, $0b, $09, $0b, $09, $0b, $0b, $0b, $0b, $0b, $0b, $0b,
$8b, $06
    .byte   $04, $03, $02, $84
    .byte $13, $13, $0e, $0b, $09, $09, $09, $0b, $09, $09, $09, $89
    .byte $1d, $1d, $1d, $1d, $1d, $1d, $1d, $1d, $1d, $1a, $1a, $19, $19, $19,
$19, $19
    .byte   $19, $1d, $1d, $1d, $1d, $1d, $14, $15, $14, $15, $14, $15, $14, $15,
$14, $15
    .byte   $14, $95
    .byte $18, $19, $1a, $1b, $1c, $1d, $1e, $9f

QuickSandTab:
    .byte %00000000 ; |        |
    .byte %00001111 ; |    XXXX|
    .byte %00001111 ; |    XXXX|
    .byte %00000000 ; |        |
    .byte %00001111 ; |    XXXX|                next byte (0) overlaps

LadderTab:
    .byte BLACKPIT|DISABLE, BLACKPIT|DISABLE, BLACKPIT|ENABLE, BLUEPIT|ENABLE
    .byte BLUEPIT |DISABLE, BLACKPIT|DISABLE, BLACKPIT|ENABLE, BLUEPIT|ENABLE
    .byte DISABLE, DISABLE, ENABLE, ENABLE  ; some bytes overlap
LianaTab:
    .byte DISABLE, DISABLE, ENABLE, ENABLE, ENABLE, DISABLE, ENABLE, DISABLE

CheckTreasures:
    lda    random           ; 3
    rol                     ; 2
    rol                     ; 2
    rol                     ; 2
    and    #$03             ; 2
    tax                     ; 2                bits 7 & 8
    ldy    objectType       ; 3
    lda    TreasureMask,y   ; 4
    tay                     ; 2
    and    treasureBits,x   ; 4
    php                     ; 3
    tya                     ; 2
    ora    treasureBits,x   ; 4
    plp                     ; 4
```

```
    rts                     ; 6

SetDigitPtrs SUBROUTINE
    ldx    #2               ; 2                 -> 8
.loopSet:
    txa                     ; 2
    asl                     ; 2
    asl                     ; 2
    tay                     ; 2
    jsr    BCD2DigitPtrs    ; 6
    dex                     ; 2
    bpl    .loopSet         ; 2
    rts                     ; 6

KilledHarry SUBROUTINE
    lda    #SOUND_DEAD      ; 2
    sta    soundIdx         ; 3
    lda    #$84             ; 2                 start copyright..
    sta    noGameScroll     ; 3                 ..animation
    jmp    ProcessObjects   ; 3

; the bounds of the holes and pits where Harry falls down:
HoleBoundsTab:
    .byte 72, 79,   0,  0,   0,  0,   0,  0    ; single hole
    .byte 44, 55,  72, 79,  96,107,   0,  0    ; triple hole
    .byte 44,107,   0,  0,   0,  0,   0,  0    ; pit
    .byte 44, 55,  64, 71,  80, 87,  96,107    ; closed croco jaws
    .byte 44, 61,  64, 77,  80, 93,  96,107    ; open croco jaws

    align 256

Log0:
    .byte %00000000; |        |
    .byte %00011000; |   XX   |
    .byte %00100100; |  X  X  |
    .byte %01011010; | X XX X |
    .byte %01011010; | X XX X |
    .byte %01011010; | X XX X |
    .byte %01100110; | XX  XX |
    .byte %01111110; | XXXXXX |
    .byte %01011110; | X XXXX |
    .byte %01110110; | XXX XX |
    .byte %01111110; | XXXXXX |
    .byte %01011110; | X XXXX |
    .byte %01110110; | XXX XX |
    .byte %00111100; |  XXXX  |
    .byte %00011000; |   XX   |
    .byte %00000000; |        |
Log1:
    .byte %00000000; |        |
    .byte %00011000; |   XX   |
    .byte %00100100; |  X  X  |
    .byte %01011010; | X XX X |
    .byte %01011010; | X XX X |
    .byte %01011010; | X XX X |
    .byte %01100110; | XX  XX |
    .byte %01111110; | XXXXXX |
    .byte %01111010; | XXXX X |
```

```
    .byte %01101110; | XX XXX |
    .byte %01111110; | XXXXXX |
    .byte %01111010; | XXXX X |
    .byte %01101110; | XX XXX |
    .byte %00111100; |  XXXX  |
    .byte %00011000; |   XX   |
    .byte %00000000; |        |
Fire0:
    .byte %00000000; |        |
    .byte %11000011; |XX    XX|
    .byte %11100111; |XXX  XXX|
    .byte %01111110; | XXXXXX |
    .byte %00111100; |  XXXX  |
    .byte %00011000; |   XX   |
    .byte %00111100; |  XXXX  |
    .byte %01111100; | XXXXX  |
    .byte %01111100; | XXXXX  |
    .byte %01111000; | XXXX   |
    .byte %00111000; |  XXX   |
    .byte %00111000; |  XXX   |
    .byte %00110000; |  XX    |
    .byte %00110000; |  XX    |
    .byte %00010000; |   X    |
    .byte %00010000; |   X    |
Fire1:
    .byte %00000000; |        |
    .byte %11000011; |XX    XX|
    .byte %11100111; |XXX  XXX|
    .byte %01111110; | XXXXXX |
    .byte %00111100; |  XXXX  |
    .byte %00011000; |   XX   |
    .byte %00111100; |  XXXX  |
    .byte %00111110; |  XXXXX |
    .byte %00111110; |  XXXXX |
    .byte %00011110; |   XXXX |
    .byte %00011100; |   XXX  |
    .byte %00011100; |   XXX  |
    .byte %00001100; |    XX  |
    .byte %00001100; |    XX  |
    .byte %00001000; |    X   |
    .byte %00001000; |    X   |
Cobra0:
    .byte %00000000; |        |
    .byte %11111110; |XXXXXXX |
    .byte %11111001; |XXXXX  X|
    .byte %11111001; |XXXXX  X|
    .byte %11111001; |XXXXX  X|
    .byte %11111001; |XXXXX  X|
    .byte %01100000; | XX     |
    .byte %00010000; |   X    |
    .byte %00001000; |    X   |
    .byte %00001100; |    XX  |
    .byte %00001100; |    XX  |
    .byte %00001000; |    X   |
    .byte %00111000; |  XXX   |
    .byte %00110000; |  XX    |
    .byte %01000000; | X      |
    .byte %00000000; |        |
```

```
Cobra1:
    .byte %00000000; |        |
    .byte %11111110; |XXXXXXX |
    .byte %11111001; |XXXXX  X|
    .byte %11111001; |XXXXX  X|
    .byte %11111010; |XXXXX X |
    .byte %11111010; |XXXXX X |
    .byte %01100000; | XX     |
    .byte %00010000; |   X    |
    .byte %00001000; |    X   |
    .byte %00001100; |    XX  |
    .byte %00001100; |    XX  |
    .byte %00001000; |    X   |
    .byte %00111000; |  XXX   |
    .byte %00110000; |  XX    |
    .byte %10000000; |X       |
    .byte %00000000; |        |
Croco0:
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %11111111; |XXXXXXXX|
    .byte %10101011; |X X X XX|
    .byte %00000011; |      XX|
    .byte %00000011; |      XX|
    .byte %00001011; |    X XX|
    .byte %00101110; |  X XXX |
    .byte %10111010; |X XXX X |
    .byte %11100000; |XXX     |
    .byte %10000000; |X       |
    .byte %00000000; |        |
    .byte %00000000; |        |
Croco1:
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %11111111; |XXXXXXXX|
    .byte %10101011; |X X X XX|
    .byte %01010101; | X X X X|
    .byte %11111111; |XXXXXXXX|
    .byte %00000110; |     XX |
    .byte %00000100; |     X  |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
MoneyBag:
    .byte %00000000; |        |
    .byte %00111110; |  XXXXX |
    .byte %01110111; | XXX XXX|
    .byte %01110111; | XXX XXX|
    .byte %01100011; | XX   XX|
    .byte %01111011; | XXXX XX|
```

```
    .byte %01100011; | XX   XX|
    .byte %01101111; | XX XXXX|
    .byte %01100011; | XX   XX|
    .byte %00110110; |  XX XX |
    .byte %00110110; |  XX XX |
    .byte %00011100; |   XXX  |
    .byte %00001000; |    X   |
    .byte %00011100; |   XXX  |
    .byte %00110110; |  XX XX |
    .byte %00000000; |        |
Scorpion0:
    .byte %10000101; |X    X X|
    .byte %00110010; |  XX  X |
    .byte %00111101; |  XXXX X|
    .byte %01111000; | XXXX   |
    .byte %11111000; |XXXXX   |
    .byte %11000110; |XX   XX |
    .byte %10000010; |X     X |
    .byte %10010000; |X  X    |
    .byte %10001000; |X   X   |
    .byte %11011000; |XX XX   |
    .byte %01110000; | XXX    |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
Scorpion1:
    .byte %01001001; | X  X  X|
    .byte %00110011; |  XX  XX|
    .byte %00111100; |  XXXX  |
    .byte %01111000; | XXXX   |
    .byte %11111010; |XXXXX X |
    .byte %11000100; |XX   X  |
    .byte %10010010; |X  X  X |
    .byte %10001000; |X   X   |
    .byte %11011000; |XX XX   |
    .byte %01110000; | XXX    |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
Wall:
    .byte %11111110; |XXXXXXX |
    .byte %10111010; |X XXX X |
    .byte %10111010; |X XXX X |
    .byte %10111010; |X XXX X |
    .byte %11111110; |XXXXXXX |
    .byte %11101110; |XXX XXX |
    .byte %11101110; |XXX XXX |
    .byte %11101110; |XXX XXX |
    .byte %11111110; |XXXXXXX |
    .byte %10111010; |X XXX X |
    .byte %10111010; |X XXX X |
    .byte %10111010; |X XXX X |
    .byte %11111110; |XXXXXXX |
```

```
    .byte %11101110; |XXX XXX |
    .byte %11101110; |XXX XXX |
    .byte %11101110; |XXX XXX |
Bar0:
    .byte %00000000; |        |
    .byte %11111000; |XXXXX   |
    .byte %11111100; |XXXXXX  |
    .byte %11111110; |XXXXXXX |
    .byte %11111110; |XXXXXXX |
    .byte %01111110; | XXXXXX |
    .byte %00111110; |  XXXXX |
    .byte %00000000; |        |
    .byte %00010000; |   X    |
    .byte %00000000; |        |
    .byte %01010100; | X X X  |
    .byte %00000000; |        |
    .byte %10010010; |X  X  X |
    .byte %00000000; |        |
    .byte %00010000; |   X    |
    .byte %00000000; |        |
Bar1:
    .byte %00000000; |        |
    .byte %11111000; |XXXXX   |
    .byte %11111100; |XXXXXX  |
    .byte %11111110; |XXXXXXX |
    .byte %11111110; |XXXXXXX |
    .byte %01111110; | XXXXXX |
    .byte %00111110; |  XXXXX |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00101000; |  X X   |
    .byte %00000000; |        |
    .byte %01010100; | X X X  |
    .byte %00000000; |        |
    .byte %00010000; |   X    |
    .byte %00000000; |        |
    .byte %00000000; |        |
Ring:
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00111000; |  XXX   |
    .byte %01101100; | XX XX  |
    .byte %01000100; | X   X  |
    .byte %01000100; | X   X  |
    .byte %01000100; | X   X  |
    .byte %01101100; | XX XX  |
    .byte %00111000; |  XXX   |
    .byte %00010000; |   X    |
    .byte %00111000; |  XXX   |
    .byte %01111100; | XXXXX  |
    .byte %00111000; |  XXX   |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
Nothing:
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
```

```
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |

    align 256

  IF OPTIMIZE
LogColor:
    .byte BROWN, BROWN, BROWN, BROWN, BROWN, BROWN, BROWN, BROWN
    .byte BROWN, BROWN, BROWN, BROWN, BROWN, BROWN, BROWN
    FILL_NOP 1         ; JTZ: the logs are only 15 lines tall
.pcFire:
FireColor = .pcFire - 1
    .byte BROWN-2, BROWN-2, BROWN-2, BROWN-2, ORANGE, ORANGE, ORANGE
    .byte $2e, $2e, $2e, $2e, $2e, $2e, $2e
    FILL_NOP 2         ; JTZ: the fire is only 14 lines tall
.pcCobra:
CobraColor = .pcCobra-1
    .byte BLACK, GREY-2, BLACK, GREY-2, BLACK, BLACK, BLACK
    .byte BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, DARK_RED
    FILL_NOP 2         ; JTZ: the cobra is only 14 bytes tall
.pcCroco:
CrocoColor = .pcCroco-5
    .byte DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2,
DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2
    .byte DARK_GREEN-2
    FILL_NOP 7         ; JTZ: the crocos are only 9 lines tall
.pcMoneyBag:
MoneyBagColor = .pcMoneyBag-1
    .byte GREY-2, GREY-2, GREY-2, GREY-2, GREY-2, GREY-2, GREY-2
    .byte GREY-2, GREY-2, GREY-2, GREY-2, BROWN, GREY-2, GREY-2
    FILL_NOP 2         ; JTZ: the moneybag is only 9 lines tall
ScorpionColor:
    .byte WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE
    .byte WHITE, WHITE, WHITE
     FILL_NOP 5         ; JTZ: the scorpion is only 11 lines tall
  ELSE
LogColor:
    .byte BROWN, BROWN, BROWN, BROWN, BROWN, BROWN, BROWN, BROWN
    .byte BROWN, BROWN, BROWN, BROWN, BROWN, BROWN, BROWN, BROWN
FireColor:
    .byte BROWN-2, BROWN-2, BROWN-2, BROWN-2, BROWN-2, ORANGE, ORANGE, ORANGE
    .byte $2e, $2e, $2e, $2e, $2e, $2e, $2e, $2e
CobraColor:
    .byte BLACK, BLACK, GREY-2, BLACK, GREY-2, BLACK, BLACK, BLACK
    .byte BLACK, BLACK, BLACK, BLACK, BLACK, BLACK, DARK_RED, DARK_RED
CrocoColor:
    .byte DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2,
```

```
DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2
    .byte DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2,
DARK_GREEN-2, DARK_GREEN-2, DARK_GREEN-2
MoneyBagColor:
    .byte GREY-2, GREY-2, GREY-2, GREY-2, GREY-2, GREY-2, GREY-2, GREY-2
    .byte GREY-2, GREY-2, GREY-2, GREY-2, BROWN, GREY-2, GREY-2, GREY-2
ScorpionColor:
    .byte WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE
    .byte WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE
  ENDIF
WallColor:
    .byte GREY, DARK_RED, DARK_RED, DARK_RED, GREY, DARK_RED, DARK_RED, DARK_RED
    .byte GREY, DARK_RED, DARK_RED, DARK_RED, GREY, DARK_RED
RingColor:
    .byte DARK_RED, DARK_RED
GoldBarColor:
    .byte YELLOW, YELLOW, YELLOW, YELLOW, YELLOW, YELLOW, YELLOW, YELLOW
    .byte WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE
    .byte WHITE
SilverBarColor:
    .byte GREY, GREY, GREY, GREY, GREY, GREY, GREY, WHITE
    .byte WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE, WHITE

InitTab:
    .word Harry5                ; harryPatPtr
    .word Log0                  ; objPatPtr
    .word RunColTab             ; harryColPtr
    .word LogColor              ; objColPtr
    .word Wall
    .word WallColor
    .word Wall
    .word WallColor
    .word Space
    .word Space
    .word Two
    .word Zero
    .word Zero
    .word Zero


RightRandom SUBROUTINE
; generate new random scene on the right:
.loopRandom:
; random' = random << 1 | (bit3^bit4^bit5^bit7)
    lda    random           ; 3
    asl                     ; 2
    eor    random           ; 3
    asl                     ; 2
    eor    random           ; 3
    asl                     ; 2
    asl                     ; 2
    eor    random           ; 3
    asl                     ; 2
    rol    random           ; 5
    dex                     ; 2
    bpl    .loopRandom      ; 2
    jmp    ContRandom       ; 3
```

```
; low pointer to harry data:
HarryPtrTab:
    .byte <Harry0
    .byte <Harry1
    .byte <Harry2
    .byte <Harry3
    .byte <Harry4
    .byte <Harry5
    .byte <Harry6
    .byte <Harry7
    .byte <Harry8

JumpTab:
; increase/decrease y-position of jumping Harry:
    .byte  1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1
    .byte -1, 0, 0, 0,-1, 0, 0,-1, 0,-1,-1,-1,-1,-1,-1,-1

ObjectPtrTab:
    .byte <Log0
    .byte <Log0
    .byte <Log0
    .byte <Log0
    .byte <Log0
    .byte <Log0
    .byte <Fire0
    .byte <Cobra0
    .byte <MoneyBag
    .byte <Bar0         ; silver bar
    .byte <Bar0         ; gold bar
    .byte <Ring
    .byte <Nothing

TreasureMask:
    .byte %10000000
    .byte %01000000
    .byte %00100000
    .byte %00010000
    .byte %00001000
    .byte %00000100
    .byte %00000010
    .byte %00000001

    align 256

Zero:
    .byte %00111100; |  XXXX  |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %00111100; |  XXXX  |
One:
    .byte %00111100; |  XXXX  |
    .byte %00011000; |   XX   |
    .byte %00011000; |   XX   |
    .byte %00011000; |   XX   |
```

```
    .byte %00011000; |   XX   |
    .byte %00011000; |   XX   |
    .byte %00111000; |  XXX   |
    .byte %00011000; |   XX   |
Two:
    .byte %01111110; | XXXXXX |
    .byte %01100000; | XX     |
    .byte %01100000; | XX     |
    .byte %00111100; |  XXXX  |
    .byte %00000110; |     XX |
    .byte %00000110; |     XX |
    .byte %01000110; | X   XX |
    .byte %00111100; |  XXXX  |
Three:
    .byte %00111100; |  XXXX  |
    .byte %01000110; | X   XX |
    .byte %00000110; |     XX |
    .byte %00001100; |    XX  |
    .byte %00001100; |    XX  |
    .byte %00000110; |     XX |
    .byte %01000110; | X   XX |
    .byte %00111100; |  XXXX  |
Four:
    .byte %00001100; |    XX  |
    .byte %00001100; |    XX  |
    .byte %00001100; |    XX  |
    .byte %01111110; | XXXXXX |
    .byte %01001100; | X  XX  |
    .byte %00101100; |  X XX  |
    .byte %00011100; |   XXX  |
    .byte %00001100; |    XX  |
Five
    .byte %01111100; | XXXXX  |
    .byte %01000110; | X   XX |
    .byte %00000110; |     XX |
    .byte %00000110; |     XX |
    .byte %01111100; | XXXXX  |
    .byte %01100000; | XX     |
    .byte %01100000; | XX     |
    .byte %01111110; | XXXXXX |
Six:
    .byte %00111100; |  XXXX  |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %01111100; | XXXXX  |
    .byte %01100000; | XX     |
    .byte %01100010; | XX   X |
    .byte %00111100; |  XXXX  |
Seven:
    .byte %00011000; |   XX   |
    .byte %00011000; |   XX   |
    .byte %00011000; |   XX   |
    .byte %00011000; |   XX   |
    .byte %00001100; |    XX  |
    .byte %00000110; |     XX |
    .byte %01000010; | X    X |
    .byte %01111110; | XXXXXX |
```

```
Eight:
    .byte %00111100; |  XXXX  |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %00111100; |  XXXX  |
    .byte %00111100; |  XXXX  |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %00111100; |  XXXX  |
Nine:
    .byte %00111100; |  XXXX  |
    .byte %01000110; | X   XX |
    .byte %00000110; |     XX |
    .byte %00111110; |  XXXXX |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %01100110; | XX  XX |
    .byte %00111100; |  XXXX  |
DoublePoint:
    .byte %00000000; |        |
    .byte %00011000; |   XX   |
    .byte %00011000; |   XX   |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00011000; |   XX   |
    .byte %00011000; |   XX   |
    .byte %00000000; |        |
Space:
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
CopyRight0:
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %11110111; |XXXX XXX|
    .byte %10010101; |X  X X X|
    .byte %10000111; |X    XXX|
    .byte %10000000; |X       |
    .byte %10010000; |X  X    |
    .byte %11110000; |XXXX    |
    .byte %10101101; |X X XX X|
    .byte %10101001; |X X X  X|
    .byte %11101001; |XXX X  X|
    .byte %10101001; |X X X  X|
    .byte %11101101; |XXX XX X|
    .byte %01000001; | X     X|
    .byte %00001111; |    XXXX|
    .byte %00000000; |        |
CopyRight1:
    .byte %01000111; | X   XXX|
    .byte %01000001; | X     X|
    .byte %01110111; | XXX XXX|
    .byte %01010101; | X X X X|
```

```
    .byte %01110101; | XXX X X|
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %01010000; | X X    |
    .byte %01011000; | X XX   |
    .byte %01011100; | X XXX  |
    .byte %01010110; | X X XX |
    .byte %01010011; | X X  XX|
    .byte %00010001; |   X   X|
    .byte %11110000; |XXXX    |
    .byte %00000000; |        |
CopyRight2:
    .byte %00000011; |      XX|
    .byte %00000000; |        |
    .byte %01001011; | X  X XX|
    .byte %01001010; | X  X X |
    .byte %01101011; | XX X XX|
    .byte %00000000; |        |
    .byte %00001000; |    X   |
    .byte %00000000; |        |
    .byte %10111010; |X XXX X |
    .byte %10001010; |X   X X |
    .byte %10111010; |X XXX X |
    .byte %10100010; |X X   X |
    .byte %00111010; |  XXX X |
    .byte %10000000; |X       |
    .byte %11111110; |XXXXXXX |
    .byte %00000000; |        |
CopyRight3:
    .byte %10000000; |X       |
    .byte %10000000; |X       |
    .byte %10101010; |X X X X |
    .byte %10101010; |X X X X |
    .byte %10111010; |X XXX X |
    .byte %00100010; |  X   X |
    .byte %00100111; |  X  XXX|
    .byte %00000010; |      X |
    .byte %11101001; |XXX X  X|
    .byte %10101011; |X X X XX|
    .byte %10101111; |X X XXXX|
    .byte %10101101; |X X XX X|
    .byte %11101001; |XXX X  X|
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
CopyRight4:
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00010001; |   X   X|
    .byte %00010001; |   X   X|
    .byte %00010111; |   X XXX|
    .byte %00010101; |   X X X|
    .byte %00010111; |   X XXX|
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
```

```
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %00000000; |        |
CopyRight5:
    .byte %00000000; |        |
    .byte %00000000; |        |
    .byte %01110111; | XXX XXX|
    .byte %01010100; | X X X  |
    .byte %01110111; | XXX XXX|
    .byte %01010001; | X X   X|
    .byte %01110111; | XXX XXX|
    .byte %00000000; |        |

ColorTab:
    .byte $0C                       ; score and copyright color
    .byte $0C                       ; unused
    .byte DARK_GREEN                ; leaves color
    .byte GREEN                     ; jungle color
    .byte BLACK                     ; hole, background and tar pit color
    .byte BROWN-2                   ; branches and log color
    .byte BROWN+2                   ; underground color
    .byte YELLOW-6                  ; ground color
    .byte BLUE                      ; swamp color

Color1PtrTab:
    .byte <LogColor
    .byte <LogColor
    .byte <LogColor
    .byte <LogColor
    .byte <LogColor
    .byte <LogColor
    .byte <FireColor
    .byte <CobraColor
    .byte <MoneyBagColor
    .byte <SilverBarColor
    .byte <GoldBarColor+1
    .byte <RingColor

NuSize1Tab:
    .byte ONE_COPY                  ; one rolling logs
    .byte TWO_COPIES                ; two rolling logs
    .byte TWO_WIDE_COPIES           ; two wide rolling logs
    .byte THREE_MED_COPIES          ; three rolling logs
    .byte ONE_COPY                  ; one stationary log
    .byte THREE_MED_COPIES          ; three stationary logs
    .byte ONE_COPY                  ; fire
    .byte ONE_COPY                  ; cobra
    .byte ONE_COPY                  ; money bag
    .byte ONE_COPY                  ; silver bar
    .byte ONE_COPY                  ; gold bar
    .byte ONE_COPY                  ; ring

; used to animate some of the hazards:
AnimateTab:
    .byte 0                         ; logs
    .byte 0
```

```
    .byte 0
    .byte 0
    .byte 0
    .byte 0
    .byte OBJECT_H                  ; fire
    .byte OBJECT_H                  ; cobra
    .byte 0                         ; money bag
    .byte OBJECT_H                  ; silver bar
    .byte OBJECT_H                  ; gold bar
    .byte 0                         ; ring
    .byte 0                         ; nothing (treasure collected)

GroundTypeTab:
    .byte <[OneHole    - PF2PatTab] ; one hole
    .byte <[ThreeHoles - PF2PatTab] ; three holes
    .byte <[Pit        - PF2PatTab] ; tar pit
    .byte <[Pit        - PF2PatTab] ; swamp
    .byte <[Pit        - PF2PatTab] ; swamp with crocodiles
    .byte $80                       ; black quicksand with treasure
    .byte $80                       ; black quicksand
    .byte $80                       ; blue quicksand

; crocodile in scene?
CrocoTab:
    .ds   CROCO_SCENE, 0
    .byte 1
    .ds   7-CROCO_SCENE, 0


    org     $fffc

    .word $f000
    .word 0
```