```
; Combat for Atari by Larry Wagner
;
; Original disassembly by Harry Dodgson
; Commented further by Nick Bensema (1997)
; Major overhaul by Roger Williams (2002)
;
; My intent in overhauling this classic disassembly is to finish it
; so that the purpose of every instruction, memory location, and
; table is made completely clear.
;
; For some reason the NBCOMBAT file ORG statements all point to
; the region $1000-$1FFF; this would play in a VCS but the cartridge
; .BIN I have is mapped from $F000-$FFFF.  This file compiles with
; DASM to an image which differs from this ROM only in the few
; unwritten bytes between the end of data and the startup vectors.
; DASM sets these to zero, typical of unwritten RAM, but in the cart
; they are $FF, typical of unprogrammed PROM.
;
; Thanks to Brian Prescott for pointing me to Joe DeCuir's
; presentation notes, which revealed Atari's original names
; for the main loop toplevel routines and offered some guidance
; on their separation of function.
;
; I have removed some of the breathless intro-to-VCS and historical
; comments.  This version assumes a basic familiarity with VCS
; programming, and is meant as a basis for hacking the COMBAT game
; itself.  There are plenty of resources outside of this file if
; you don't know how the VCS works.
;
; For reference, as this is rather important when reading the,
; code, here is the game variation matrix (it is not trivially
; obvious how this corresponds to GAMVAR):
;
;                Game No.                Open Field
;                |  Straight Missiles    |  Easy Maze
;                |  |  Guided Missiles   |  |  Complex Maze
;                |  |  |  Machine Guns   |  |  |  Clouds
;                |  |  |  |  Direct Hit  |  |  |  |
;                |  |  |  |  |  Billiard |  |  |  |
;                |  |  |  |  |  |   Hit  |  |  |  |
;                |  |  |  |  |  |        |  |  |  |
;
;TANK            1  -  X  -  -  -        X  -  -  -
;                2  -  X  -  -  -        -  X  -  -
;                3  X  -  -  -  -        -  X  -  -
;                4  -  X  -  -  -        -  -  X  -
;                5  X  -  -  -  -        -  -  X  -
;-------------------------------------------------
;TANK-PONG       6  -  -  -  X  X        -  X  -  -
;                7  -  -  -  X  X        -  -  X  -
;                8  -  -  -  -  X        X  -  -  -
;                9  -  -  -  -  X        -  X  -  -
;-------------------------------------------------
;INVISIBLE TANK 10  -  X  -  -  -        X  -  -  -
;               11  -  X  -  -  -        -  X  -  -
;-------------------------------------------------
;INVISIBLE      12  -  -  -  X  X        -  X  -  -
;TANK-PONG      13  -  -  -  -  X        X  -  -  -
;               14  -  -  -  -  X        -  X  -  -
;-------------------------------------------------
;BI-PLANE       15  -  X  -  -  -        -  -  -  X
;               16  X  -  -  -  -        -  -  -  X
;               17  -  -  X  -  -        -  -  -  X
;               18  -  -  X  -  -        X  -  -  -
;     2 vs. 2   19  -  X  -  -  -        X  -  -  -
;     1 vs. 3   20  X  -  -  -  -        X  -  -  -
;-------------------------------------------------
;JET            21  -  X  -  -  -        -  -  -  X
;               22  X  -  -  -  -        -  -  -  X
;               23  -  X  -  -  -        X  -  -  -
;               24  X  -  -  -  -        X  -  -  -
;     2 vs. 2   25  -  X  -  -  -        -  -  -  X
;     1 vs. 3   26  -  X  -  -  -        X  -  -  -
;     2 vs. 2   27  X  -  -  -  -        X  -  -  -


        processor 6502
        include vcs.h

; RAM is cleared in blocks beginning at various addresses and
; always ending at $A2 (though this isn't the highest address
; used).  I have placed \\\/// comments to mark these points.

BINvar  =     $80 ; Master Game Variation Control (binary)
                  ; (When BINvar is reset or incremented,
                  ; BCDvar is reset or BCD-imcremented and
                  ; GAMVAR flag is read from VARMAP+BINvar)
BCDvar  =     $81 ; Game Variation in BCD
;
;\\\///
;
; $82 thru $85 contain flags built from GAMVAR for quick testing via BIT.
;
PF_PONG =     $82 ; bit 7 DIS-able playfield flag
                  ; bit 6 Pong missiles (bounce off playfield)
GUIDED  =     $83 ; bit 7 = guided missile game
                  ; bit 6 = machine gun game
BILLIARD =    $84 ; Just bit 6 = billiard hit game (missiles can't
                  ; hit tank until at least 1 bounce off playfield)
GAMSHP  =     $85 ; Shape of player and game type
                  ; 0 = Tank
                  ; 1 = Biplane
                  ; 2 = Jet Fighter
;
CLOCK   =     $86 ; Master timer inc'd every frame during VSYNC
                  ; in NBCOMBAT this was misleadingly labelled GTIMER
SHOWSCR =     $87 ; Show/hide RIGHT player score (left only is used
                  ; to indicate game selection in attract mode)  To
                  ; inhibit both scores, KLskip is set to $0E vs. $02
GameOn  =     $88 ; $00=attract mode, $FF=game going on.  Bits 7, 1,
                  ; and "all" tested in various places.  Incorrectly set
                  ; to $10 at START, but must not be a problem :-)
;\\\///
;
SelDbnce =    $89 ; Select Switch Debounce flag which prevents a
                  ; hold-down from registering as 60 presses/second
StirTimer =   $8A ; Bit 0 = identity of loser during tank stir
                  ; Bits 2-7 = countdown timer controlling stir after loss
Vtemp     =   $8B ; Temp storage for current velocity
FwdTimer  =   $8D ; FwdTimer must count $F0 to $00 between changes in
        thru $8E ; forward motion control; also used for momentum pacing
                 $8F ; ...
        thru $90 ; seem to be reserved too (missiles?) but not used
LastTurn =    $91 ; Flag indicating direction of last turn, used
        thru $92 ; to inhibit whipsaw direction changes (may
                 ; have been intended for rotational momentum)
TurnTimer =   $93 ; Countdown timer between 22.5-degree rotates
        thru $94 ; for P0 and P1
DIRECTN =     $95 ; Players and missiles' current bearing.
        thru $98 ; (4 bytes P0,P1,M0,M1)
MisLife =     $99 ; Missile Lifetime down-counters
        thru $9A
BounceCount = $9B ; (1) Billiard bounced-once flag, via any value other
        thru $9C ; than $1F init value; (2) Pong sound tone freq, which
                 ; ascends in tone as BounceCount DECed with each bounce
MxPFcount =   $9D ; During Pong bounce, count of collision duration in
        thru $9E ; frames, used to try different heading adjustments
                 ; until "desired" reflection achieved
AltSnd  =     $9F ; Alt Player Sound flag/counter; 0=normal motor sound,
        thru $A0 ; else counts up to $04 to time Pong sound
SCORE   =     $A1 ; Player scores in BCD.
        thru $A2 ;
;
;\\\/// Addresses beyond here aren't ever cleared by ClearMem.
;
GAMVAR  =     $A3 ; Game Variation bitwise descriptor via VARMAP
TankY0  =     $A4 ; Tank 0's Y-position
TankY1  =     $A5 ; and tank 1
MissileY0 =   $A6 ; Missile 0's Y-position
MissileY1 =   $A7 ; and missile 1
MVadjA  =     $A8 ; First-half FwdTimer-Velocity adjustments
        thru $A9 ; for each player.  By an amazing coincidence
                 ; in all games these seem to be the same as
                 ; the *current* velocity.
MVadjB  =     $AA ; Second-half FwdTimer-Velocity adjustments,
        thru $AB ; which seem to be the same as the *final* velocity.
MPace   =     $AC ; Pacing counter; never initialized!  INC'd and
                  ; masked to pace certain actions slower than
        thru $AF ; once/frame, for each player & missile
XOFFS   =     $B0 ; X-offset for pending Hmove.
XoffBase =    $B1 ; $0, $10, $20, or $30 offset into X-offset tbl
OldMisDir =   $B2 ; Missile bearing before a Pong-bounce began
        thru $B3 ;
ScanLine =    $B4 ; Current scanline on the playfield.
LORES   =     $B5 ; lo-res indirect addresses.
        thru $BA ; 6 bytes / 3 16-bit pointers
SHAPES  =     $BB ; Pointer to player sprites
HIRES   =     $BD ; Hi-res (sprite) shape buffer.  Left player's shape
        thru $CC ; stored in even bytes, right player's in odd.
TEMP1   =     $D1 ; Temp storage for several quick save/math operations
TEMP    =     $D2 ; "score conversion temporary"
TMPSTK  =     $D3 ; Temporary storage for stack.
DIFSWCH =     $D5 ; Hold & shift temp for console switches
Color0  =     $D6 ; Colors loaded from ColorTbl for player 0 and 1
Color1  =     $D7 ; These may be changed e.g. invisible tanks
XColor0 =     $D8 ; Repeated P0 and P1 Colors for reference, used
XColor1 =     $D9 ; to restore ColorX after a change
ColorPF =     $DA ; BK and PF colors loaded in same block as XColorX.
ColorBK =     $DB ; Never changed, so no reference versions are kept.
KLskip  =     $DC ; Kernal lines to skip before score, or main w/o score
                  ; (Also used in Kernal as flag whether to show score)
GameTimer =   $DD ; Master game timer set to $80 when game starts,
                  ; incremented until overflow at $FF-->$00 ends game
                  ; Bit 7 indicates game in play, also used w/GameOn to
                  ; flash score.  During attract mode GameTimer is used
                  ; to cycle colors; this is OK since it only assumes
                  ; its game-timing function if GameOn != $00.
NUMG0   =     $DE ; Storage for current byte
NUMG1   =     $DF ; of score number graphics.
SCROFF  =     $E0 ; Score pattern offsets (4 bytes)
        thru $E3 ; lo nibble 0, lo 1, hi 0, hi 1
COLcount =    $E4 ; Counter keeps tank-tank and tank-PF collisions from
        thru $E5 ; affecting a stationary tank's bearing unless the
                 ; collision lasts at least 4 cycles
;
StkTop  =     $FF ; Top of stack (which IS used, at least 8 bytes)
;
; So much for the RAM.  Here's the ROM:

org $F000

STARTSEI                  ; Disable interrupts
CLD                     ; Clear decimal bit
LDX  #StkTop
TXS                     ; Init Stack
LDX  #$5D
JSR  ClearMem           ; zero out RAM except address $A2
LDA  #$10;
STA  SWCHB+1            ; Port B data direction register and
STA  GameOn             ; GameOn (tho not quite a valid value)...
JSR  ClrGam; clear game RAM $82-$A2
;
MLOOP   JSR  VCNTRL             ; Generate a VSYNC and begin VBLANK
;
; VBLANK logic:
;
JSR  GSGRCK             ; Parse console switches
JSR  LDSTEL             ; Load Stella Registers
JSR  CHKSW              ; Check Joystick Switches
JSR  COLIS              ; Check Collision Registers
JSR  STPMPL             ; Setup Player, Missile Motion
JSR  ROT                ; Rotate Sprites
JSR  SCROT              ; Calculate Score Offsets
;
JSR  VOUT               ; do the Kernal (trashes the stack ptr,
                        ; but then restores it because it IS
JMP  MLOOP              ; used when we reiterate this loop)
;
; ---------------------------------------------------------------
;
; Vertical CoNTRoL
;
; Vertical sync, basic frame-start housekeeping
;
VCNTRL  INC  CLOCK              ; Master frame count timer
STA  HMCLR            ; Clear horizontal move registers.
LDA  #2              ; Get this ready...
STA  WSYNC            ; for start of next line...
STA  VBLANK           ; Start vertical blank.
STA  WSYNC
STA  WSYNC            ; and do three lines
STA  WSYNC
STA  VSYNC            ; Now start vertical sync
STA  WSYNC
STA  WSYNC            ; and do three lines
LDA  #0               ; get this ready
STA  WSYNC
STA  VSYNC            ; End of vertical sync pulse
LDA  #43              ; And set VBLANK timer
STA  TIM64T           ; with 64 clock interval.
RTS
;
; ---------------------------------------------------------------
;
; Video OUT -- THE KERNAL
;
; We start with the score, then we render the playfield, players,
; and missiles simultaneously. All in all, an average day for a VCS.
;
VOUTLDA  #$20
STA  ScanLine           ; We're assuming scanline $20.
STA  WSYNC
STA  HMOVE              ; Move sprites horizontally.
VOUT_VBLDA  INTIM
BNE  VOUT_VB            ; Wait for INTIM to time-out.
STA  WSYNC
STA  CXCLR              ; Clear collision latches
STA  VBLANK             ; End vertical blank
TSX
STX  TMPSTK             ; Save stack pointer
LDA  #$02
STA  CTRLPF             ; Double, instead of reflect.
LDX  KLskip
Vskip1  STA  WSYNC              ; Skip a few scanlines...
DEX
BNE  Vskip1
LDA  KLskip
CMP  #$0E               ; "No Score" value of KLskip
BEQ  Vmain
;
; KLskip is set as such so that when the score is
; to be displayed, it waits for just the right time
; to start drawing the score, but if the score is
; not to be displayed, as when the score flashes
; signifying "time's almost up", it waits for just
; the right time to start drawing the rest of the
; screen.
```

```
;
; Draw the score:
;
LDX  #$05              ; Score is five bytes high.
LDA  #$00              ; Clear number graphics.
STA  NUMG0             ; They won't be calculated yet,
STA  NUMG1             ; but first time through the loop
                ; the game will try to draw with
                ; them anyway.
VSCORSTA  WSYNC        ; Start with a fresh scanline.
LDA  NUMG0             ; Take last scanline's left score,
STA  PF1               ; and recycle it,
;
; Here, we begin drawing the next scanline's
; left score, as the electron beam moves towards
; the right score's position in this scanline.
;
LDY  SCROFF+2
LDA  NUMBERS,Y         ; Get left digit.
AND  #$F0
STA  NUMG0
LDY  SCROFF
LDA  NUMBERS,Y         ; Get right digit.
AND  #$0F
ORA  NUMG0
STA  NUMG0             ; Left score is ready to ship.
LDA  NUMG1             ; Take last scanline's right score,
STA  PF1               ; and recycle it.
LDY  SCROFF+3
LDA  NUMBERS,Y         ; Left digit...
AND  #$F0
STA  NUMG1
LDY  SCROFF+1
LDA  NUMBERS,Y         ; right digit...
AND  SHOWSCR
;
; Now, we use our fresh, new score graphics in this next scanline.
;
STA  WSYNC                      ; *COUNT*
ORA  NUMG1   ;Finish calculating    (0) +3
STA  NUMG1   ;right score.          (3) +3
LDA  NUMG0                      ; (6) +3
STA  PF1                        ; *9* +3
;
; We use this time to check whether we're at the end of our loop.
;
DEX                             ; (12)+2
BMI  Vmain                      ; (14)+2 No Branch
;
; If so, we're out of here.  Don't worry, the score will be
; cleared immediately, so nobody will know that we've gone
; past five bytes and are displaying garbage.
;
INC  SCROFF                     ; (16)+5
INC  SCROFF+2          ; Get ready to draw the next
INC  SCROFF+1          ; line of the byte.
INC  SCROFF+3
LDA  NUMG1
STA  PF1               ; Right score is in place.
JMP  VSCOR             ; Go to next scanline,
;
; Main Kernal Display loop for the game itself
;
VmainLDA  #$00              ; Inner Display Loop
STA  PF1               ; Clear the score.
STA  WSYNC
LDA  #$05
STA  CTRLPF            ; Reflecting playfield.
LDA  Color0
STA  COLUP0            ; How often must THIS be done?
LDA  Color1
STA  COLUP1
VfieldLDX  #$1E             ; Very Sneaky -
TXS                    ; Set stack to missile registers
SEC
;
; This yields which line of player 0 to draw.
;
LDA  TankY0
SBC  ScanLine          ; A=TankY0-ScanLine
AND  #$FE              ; Force an even number
TAX                    ; Only sixteen bytes of
AND  #$F0              ; sprite memory, so...
BEQ  VdoTank           ; If not valid,
LDA  #$00              ; blank the tank.
BEQ  VnoTank           ;       (unconditional branch)
VdoTank LDA  HIRES,X           ; Else, load the appropriate byte.
VnoTank STA  WSYNC             ; ----END OF ONE LINE----
STA  GRP0              ; Just for player 0.
;
; The infamous Combat Stack Trick:
;
; Keep in mind that at this point, the stack pointer
; is set to the missile registers, and the "zero-result"
; bit of the P register is the same at the bit ENAM0/1
; looks at.
;
LDA  MissileY1
EOR  ScanLine
AND  #$FE
PHP                    ; This turns the missle 1 on/off
LDA  MissileY0
EOR  ScanLine
AND  #$FE
PHP                    ; This turns the missle 0 on/off
;
; We've got the missile taken care of.
; Now let's see which line of the playfield to draw.
;
LDA  ScanLine
BPL  VvRefl            ; If on the bottom half of the screen,
EOR  #$F8              ; reverse direction so we can mirror.
VvReflCMP  #$20
BCC  VfDone            ; Branch if at bottom.
LSR
LSR
LSR                    ; Divide by eight,
TAY                    ; and stow it in the Y-register.
;
; By now, the electron beam is already at the next
; scanline, so we don't have to do a STA WSYNC.
;
; This yields which line of Tank 1 to draw.
;
VfDoneLDA  TankY1           ; TankY1 is other player's position.
SEC
SBC  ScanLine          ; A=TankY1 - ScanLine
INC  ScanLine          ; Increment the loop.
NOP
ORA  #$01              ; Add bit 0, force odd number.
TAX
;
AND  #$F0              ; There are only sixteen bytes of
BEQ  VdoT1             ; sprite memory, so...
LDA  #$00              ; If tank is not ready, blank it.
BEQ  VnoT1
VdoT1   LDA  HIRES,X           ; Else, draw the tank
VnoT1   BIT  PF_PONG
STA  GRP1
BMI  VnoPF             ; If PF_PONG bit 7 set, don't write PF
LDA  (LORES),Y         ; (this means game variation has blank
STA  PF0               ; background)
LDA  (LORES+2),Y
STA  PF1
LDA  (LORES+4),Y
STA  PF2
VnoPFINC  ScanLine          ; One more up in the loop.
LDA  ScanLine
EOR  #$EC              ; When we've reached the $ECth line,
BNE  Vfield            ; we've had enough.
LDX  TMPSTK            ; Restore stack pointer, which is
TXS                    ; is used for calls in main game loop
STA  ENAM0             ; Clear a bunch of registers.
STA  ENAM1
STA  GRP0
STA  GRP1
STA  GRP0              ; In case GRP0 isn't COMPLETELY zeroed.
STA  PF0
STA  PF1
STA  PF2
RTS

; ---------------------------------------------------------------
;
; Game Select Game Reset CheCK
;
; Executed immediately after VCNTRL, this subroutine parses all
; the console switches.
;
GSGRCK;
LDA  SWCHB             ; Start/Reset button....
LSR                    ; Shove bit 0 into carry flag,
BCS  NoNewGM           ; and if it's pushed...
;
; Start a new game.
;
LDA  #$0F
STA  SHOWSCR           ; Show right score.
LDA  #$FF              ; Set all bits
STA  GameOn            ; in GameOn.
LDA  #$80
STA  GameTimer         ; and bit 7 of GameTimer (this is not too
                       ; significant, as GameTimer rollover is
                       ; only checked if GameOn<>$00)
LDX  #$E6
JSR  ClearMem          ; zero out $89 thru $A2
BEQ  ResetField        ; Unconditional branch
;
NoNewGM LDY  #$02              ; Assume score to be drawn
LDA  GameTimer         ; If game in play (GameOn=$FF) AND
AND  GameOn            ; GameTimer < 7/8 finished @ $F0,
CMP  #$F0              ; draw the score unconditionally.
BCC  SCdrawn
LDA  CLOCK             ; CLOCK used to flash score near end
AND  #$30              ; of play, note the peripheral synchronization
BNE  SCdrawn           ; with GameTimer's timing of the game, which
                       ; always ends when CLOCK & $3F = 0.  CLOCK
                       ; is used here because the score blink
                       ; off duty cycle is a too quick for
                       ; GameTimer to handle, being about 1/3 sec.
LDY  #$0E              ; Set this for no score
SCdrawn STY  KLskip            ; where the Kernal will find it
LDA  CLOCK
AND  #$3F              ; CLOCK also used to slow debounce reset
BNE  ChkSel
;
; GameTimer is incremented and SelDbnce reset when
; CLOCK & $3F = 0.  This occurs 1 frame out of 64 or
; about once/second.  Thus the game is 128*64 frames
; or about 2 minutes long.
;
STA  SelDbnce          ; Reset Select Debounce Flag.  This is
                       ; what keeps incrementing the selection
                       ; if you hold Select down for a long time.
INC  GameTimer         ; increment the Main Game ~1-sec Timer.
BNE  ChkSel            ; if GameTimer rolls over,
STA  GameOn            ; zero GameOn -- game over
;
ChkSel  LDA  SWCHB             ; Select button???
AND  #$02
BEQ  SelDown
STA  SelDbnce          ; Set flag: Sel has not been down
BNE  CS_RTS            ; Unconditional branch
;
SelDown BIT  SelDbnce          ; If Sel has been down,
BMI  CS_RTS; don't select a new game.
;
INC  BINvar            ; SELECT: Go to next game.
ClrGam  LDX  #$DF              ; Clear data from current game ($82-$A2)
ClrGRST JSR  ClearMem
LDA  #$FF
STA  SelDbnce          ; Set flag: Sel has been down.
LDY  BINvar
LDA  VARMAP,Y          ; Get feature bits for this variation.
STA  GAMVAR
EOR  #$FF              ; ##FF signifies end of variations
BNE  SelGO             ; Not at end yet, set up new game
LDX  #$DD; Clear $80-$A2; resets BINvar, BCDvar
BNE  ClrGRST           ; so we start over. BNE is unconditional.
;
SelGOLDA  BCDvar; Since we have incremented BINvar, we
SED                    ; must increment BCDvar in BCD to keep
CLC                    ; it in sync. Note BCDvar is actually
ADC  #1                ; BinVar+1, since it's incremented when
STA  BCDvar            ; we reset but don't increment BINvar.
STA  SCORE             ; Display variation as score 0
CLD
BIT  GAMVAR            ; GAMSHP was reset at ClrGam...
BPL  ResetField        ; if this is a plane game,
INC  GAMSHP            ; increase GAMSHP.
BVC  ResetField        ; if this is a jet game,
INC  GAMSHP            ; increase GAMSHP further still.
;
; Branches here when game is started, too.
;
ResetField
JSR  InitPF
;
; Assuming plane game for now, we set the right player
; at a slightly higher position than the left player,
; and the position of the right player is irrelevant.
;
LDA  #50
STA  TankY1
LDA  #134
STA  TankY0
BIT  GAMVAR            ; Check to see if it is a tank game.
BMI  CS_RTS            ; Nope, bail.
                       ; It is a tank game, so
STA  TankY1            ; Right tank has same Y value,
STA  RESP1             ; and tank is at opposite side.
LDA  #$08
STA  DIRECTN+1         ; and right player faces left.
LDA  #$20
STA  HMP0
STA  HMP1
STA  WSYNC
STA  HMOVE
CS_RTSRTS


; ---------------------------------------------------------------
;
; SCoRe OffseT
;
; Convert BCD scores to score pattern offset.
; This involves the horrible, horrible implications
; involved in multiplying by five.
```

```
;
; If it weren't for the geniuses at NMOS using BCD,
; this routine would be a nightmare.
;
; This routine starts with Player 1, writes bytes 1 & 3 of
; the table, then decrements X to write bytes 0 & 2 for P0.
;
SCROTLDX  #$01
SCROT0LDA  SCORE,X
AND  #$0F                ; Lo nibble
STA  TEMP
ASL                      ; *2
ASL                      ; *4
CLC
ADC  TEMP                ; + original * 1 = original * 5
STA  SCROFF,X
LDA  SCORE,X
AND  #$F0                ; Repeat for hi nibble.  Starts *16
LSR                      ; *8
LSR                      ; *4
STA  TEMP
LSR                      ; *2
LSR                      ; *1
CLC
ADC  TEMP                ; + (*4) = original * 5
STA  SCROFF+2,X
DEX
BPL  SCROT0              ;Decrement & repeat once for P0
RTS

; ---------------------------------------------------------------
;
; SeTuP Motion for PLayers
;
; Apply horizontal and vertical motion
;
STPMPLBIT  GUIDED
BVC  STPnoMG             ; Branch if not machine gun game.
LDA  #$30                ; (Machine gun bullets move faster)
BPL  STPMG               ; Unconditional JMP.
STPnoMGLDA  #$20
STPMGSTA  XoffBase           ; $30=machine gun, $20=normal
LDX  #$03
JSR  STPM                ; Do the honors for X=3, Missile 1
DEX
JSR  STPM                ; Now X=2, M0
;
DEX                      ; Now X=1, P1; we will DEX and loop
STPnext LDA  FwdTimer,X          ; back to run this code block again
AND  #$08                ; with X=0 for P0.
LSR                      ; (to 4) This bit on means FwdTimer has
LSR                      ; (to 2) run half of the FwdTimer period
                             ; ($F0 to $FF and roll)
                         ; This bit will index MVadjA or MVadjB
STX  TEMP1               ; Player # --> TEMP1
CLC
ADC  TEMP1
TAY                      ; Player # + FwdTimer half done*2 --> Y
LDA  MVadjA,Y            ; And retrieve MVadjA or MVadjB via Y
;
SEC                      ; assume bit 7 on
BMI  STP7set             ; OK, it is
CLC                      ; whoops, backtrack
STP7set ROL                  ; carry=bit 7, now ROL; net effect is to
;                        ; rotate left inserting duplicate MSB
STA  MVadjA,Y            ; instead of original Carry, and save it
BCC  STPnoV              ; Skip next code block if bit wasn't 1
;
LDA  MPace,X             ; Tweak velocity by changing XoffBase
AND  #$01                ; but only every other time we get here
ASL
ASL
ASL
ASL
STA  XoffBase            ; XoffBase=$0 or $10 via (MPace & 1) << 4
JSR  STPM                ; Note this is where we INC MPace
STPnoV
DEX                      ; Move to _previous_ player.
BEQ  STPnext             ; Stop if about to do player -1.  :)
RTS
;
; This routine will move both tanks and missiles.
; Special cases are made for missiles, which are
; otherwise treated as players 2 and 3.
;
; It doesn't change the X register, but it does
; utilize it.
;
STPM    INC  MPace,X
LDA  DIRECTN,X
AND  #$0F
CLC
ADC  XoffBase         ; Pick table offset by game condition
TAY
LDA  Xoffsets,Y      ; X-offset by orientation.
STA  XOFFS            ; Store the default HMPV code.
BIT  PF_PONG
BVS  STPgo            ; Branch if (fast) Pong missiles
LDA  DIRECTN,X
SEC
SBC  #$02             ; If motion is near X or Y axis,
AND  #$03
BNE  STPgo            ; don't apply delay
LDA  MPace,X          ; but if very diagonal, slow a bit by
AND  #$03             ; moving only 3 of every 4 frames
BNE  STPgo            ;
LDA  #$08             ; HMPV for no motion X or Y
STA  XOFFS            ; no motion this frame
STPgo   LDA  XOFFS
;
; (This falls through, but PhMove is also called from elsewhere)
;
; Physically move a tank (0,1) or missile (2,3)
; according to the HMPV code in A
;
PhMove  STA  HMP0,X              ; Hi nibble sets HMPx horizontal motion
AND  #$0F             ; Lo nibble...
SEC
SBC  #$08             ; less 8 for 2's complement 4-bit...
STA  $D4              ; (save this offset)
CLC
ADC  TankY0,X         ; add to Y-coordinate
BIT  GAMVAR
BMI  PhNoTank         ; Branch if a plane game.
CPX  #$02
BCS  PhNoWrap         ; Branch if moving a tank player
PhNoTank
CMP  #$DB             ; Perform vertical wrap-around
BCS  PhNoWrapTop      ; branch if over top (wrap)
CMP  #$25
BCS  PhNoWrap         ; branch if over bottom (no wrap)
PhNoWrapTop
LDA  #$D9             ; Assume we wrapped bottom to top
BIT  $D4              ; Meaning offset was negative
BMI  PhNoWrap
LDA  #$28             ; Otherwise, we wrapped top to bottom
PhNoWrap
STA  TankY0,X         ; The tank/missile is moved here.
CPX  #$02
BCS  PhnoVD           ; Skip if moving a missile.
STA  VDELP0,X         ; Vertical Delay Player X...
PhnoVDRTS

; ---------------------------------------------------------------
;
; ROTate player sprites
;
; This subroutine sets up the sprite data for each player by copying
; them into sixteen bytes of RAM.
;
; The X-register starts at $0E plus player number and goes down by two
; each time through the loop, until it hits zero.  This way, after calling
; this subroutine twice, every even-numbered byte contains the left player
; shape, and every odd-numbered byte contains the right player shape.  Since
; each player is updated every two scanlines, this saves us some math.
;
; Only the first 180 degrees of rotation has been drawn into ROM.  In the
; case of the other 180 degrees, this subroutine renders a flipped version
; by doing the following:
;
; 1. It sets the TIA's reflection flag for that player, taking care of
;    the horizontal aspect rather easily.
;
; 2. It copies the bytes into memory last-to-first instead of first-to-
;    last, using the carry bit as a flag for which to do.
;
ROTLDA  #$01              ; The LO byte of CLOCK used to
AND  CLOCK            ; select alternate players on
TAX                   ; alternate frames
LDA  DIRECTN,X
STA  REFP0,X          ; Step 1 taken care of.
AND  #$0F
TAY                   ; Y = DIRECTN[X] & 0x0F.
BIT  GUIDED
BPL  ROTnoGM          ; If it's a guided missile game,
STY  DIRECTN+2,X      ; copy player bearings to missile
ROTnoGM TXA                   ; X ^= $0E,
EOR  #$0E
TAX
TYA
ASL
ASL
ASL
CMP  #$3F             ; And so step 2 begins...
CLC
BMI  ROTnoFlip        ; Branch if <180 deg.
SEC
EOR  #$47    ;The EOR sets bits 0-2, and clears bit 4
;             to subtract 180 degrees from the memory
;             pointer, too.
ROTnoFlip TAY
;
;Put all the shapes where they ought to be.
;
ROTnextLDA  (SHAPES),Y
STA  HIRES,X
BCC  ROTinc
DEY                   ; Decrement instead of increment
DEY                   ; plus cancel the upcoming INY.
ROTincINY                 ; More of step 2.
DEX
DEX                   ; X-=2.
BPL  ROTnext          ; Do for both, 1 then 0 then stop.
RTS


; ---------------------------------------------------------------
;
; CHecK joystick SWitches
;
; If we are in the interval while a loser's tank is stirring,
; he stirs and the winner freezes or goes forward.  Otherwise,
; parse the joystick inputs and move the tanks appropriately.
;
CHKSWLDA  StirTimer          ; We must dec StirTimer by 2
SEC                   ; since bit 0 is identity of
SBC  #$02             ; the stirree
BCC  NoStir           ; If no tank is exploding,
                      ; parse joystick instead.
STA  StirTimer
CMP  #$02
BCC  StirRTS          ; RTS if tank has
                      ; just finished exploding.
AND  #$01             ; Stir the LOSER's tank.
TAX
;One of these is the tank's bearings.
INC  DIRECTN,X
LDA  XColor0,X
STA  Color0,X
LDA  StirTimer
CMP  #$F7             ; We only rush the tank for a
BCC  NoStirRush       ; small part of the stir interval
JSR  RushTank
NoStirRush
LDA  StirTimer
BPL  StirRTS          ; Don't start decrementing
                      ; volume until halfway through.
LSR
LSR                   ; StirTimer scales audio volume
LSR                   ;
BoomSndSTA  AUDV0,X           ; Set explosion sound to volume in A
LDA  #$08             ; and pitch according to player X
STA  AUDC0,X
LDA  AudPitch,X
STA  AUDF0,X
StirRTSRTS
;
; Process joysticks.
;
NoStir  LDX  #$01             ; Start with P1
LDA  SWCHB            ; Console switches.
STA  DIFSWCH          ; Store switches.  Before we return
                      ; via DEX to do P0, we will ASL this
                      ; byte so difficulty bit for working
                      ; player appears in bit 7.
LDA  SWCHA            ; Joysticks. Before we return via
                      ; DEX to do P0, we will reload and
                      ; LSR this 4 times so controls for
                      ; the working player appear in the
NextPJSBIT  GameOn            ; LO nibble.
BMI  NoFreezeJS       ; Branch if game on (via bit 7).
LDA  #$FF             ; Freeze all joystick movement.
NoFreezeJS
EOR  #$FF             ; Reverse all bits
AND  #$0F             ; Keep low four bits (working player)
;
; At this point, the joystick's switches are in
; the A-register, with a bit set wherever the
; joystick is pointed.
;
; Bit 0 = up    Bit 1 = down
; Bit 2 = left  Bit 3 = right
;
STA  TEMP
LDY  GAMSHP
LDA  CtrlBase,Y       ; Account for two-dimensional array
CLC
ADC  TEMP
TAY
LDA  CTRLTBL,Y
AND  #$0F             ; Get rotation from CTRLTBL.
STA  TEMP1            ; Stash it here
BEQ  NoTurn           ; Branch if no turn.
CMP  LastTurn,X       ; If new turn is different direction
BNE  TurnReset        ; from last turn, reset the...
NoTurn  DEC  TurnTimer,X      ; ...turn pacing delay and...
BNE  DoFwdMotion      ; ...inhibit turn this interval.
```

```
TurnReset                       ; We do turn-wait counts even when
STA  LastTurn,X         ; we aren't turning, for consistency
LDA  #$0F               ; Initial countdown value to delay
STA  TurnTimer,X        ; 22.5-degree turns
;
LDA  TEMP1              ; Retrieve rotation code
CLC                     ; Turn +/- 22.5-degrees or zero,
ADC  DIRECTN,X          ; per DIRECTN
STA  DIRECTN,X
;
; For reference, we do get here every frame (~60Hz) during game.
; COMBAT does not change player speed instantaneously; it has
; an elaborate momentum system, which is just barely noticeable
; in the course of game play.
;
DoFwdMotion
INC  FwdTimer,X         ; Inc FwdTImer and if it doesn't
BMI  SkipFwdCtrl        ; roll over, don't acknowledge velocity
LDA  CTRLTBL,Y          ; changes yet
LSR
LSR
LSR
LSR                     ; Get forward velocity from CTRLTBL
;
; This is the desired _final_ velocity of the player.  If
; it is different from the player's _current_ velocity, we
; won't reach it until the end of the FwdTimer period.
;
BIT  DIFSWCH
BMI  FwdPro             ; Branch if difficulty="Pro"
                        ; (reduces A and branches back to FwdNorm)
FwdNormSTA  Vtemp,X             ; Stash velocity in Vtemp
ASL                     ; Multiply by two
TAY                     ; Stash in Y.
LDA  MVtable,Y          ; Indexed by velocity * 2, even
STA  MVadjA,X           ; V+MVtable goes to MVadjA+X
INY                     ; Why not LDA MVtable+1,Y?
LDA  MVtable,Y
STA  MVadjB,X           ; odd V+MVtable goes to MVadjB+X
LDA  #$F0               ; Initialize FwdTimer
STA  FwdTimer,X         ; (Counts up to $00 before fwd
;                       ; motion change is final)
SkipFwdCtrl
JSR  ChkVM
LDA  SWCHA              ; Joysticks..
LSR
LSR
LSR
LSR                     ; Keep bottom four bits (Left Player)
ASL  DIFSWCH            ; Use other difficulty switch.
DEX
BEQ  NextPJS
RTS
;
FwdPro  SEC                     ; Velocity is in A
SBC  GAMSHP             ; subtract 0/tank, 1/biplane, 2/jet
BPL  FwdNorm            ; Not obvious, but this is unconditional

; ---------------------------------------------------------------
;
; Check invisible tank visibility, missile lifetime expiration;
; read trigger if appropriate and launch a new missile
;
ChkVM   LDA  GAMVAR
BMI  NoInvis            ; Branch if plane game
AND  #$01               ; check also for bit 0 (invisible).
BEQ  NoInvis
LDA  ColorBK            ; Make invisible tank invisible
STA  Color0,X
NoInvisLDA  MisLife,X
BEQ  RdTrig             ; Branch if no missile in flight
LDA  XColor0,X          ; Reset tank to normal color
STA  Color0,X
LDA  MisLife,X          ; How long does missile have to go?
CMP  #$07
BCC  MisKill            ; Branch to go ahead and kill it
BIT  DIFSWCH            ; Check difficulty
BPL  MisEZ              ; If game is hard,
CMP  #$1C               ; Compare mislife to this
BCC  MisKill            ; and expire it early.
MisEZ   CMP  #$30               ; If MisLife < 30 do motor
BCC  MotMis             ; do motor, not shot sound
CMP  #$37               ; If MisLife >= 37
BCS  MisFly             ; do sliding boom sound (shot)
BIT  GUIDED
BVC  MisFly             ; Branch if machine gun.
MisKill LDA  #$00               ; Reset missile's life, killing it
STA  MisLife,X
LDA  #$FF               ; And reset its position
ResRTS  STA  RESMP0,X           ; to player.
RTS
;
; If game in progress, Read the trigger
;
RdTrig  BIT  GameOn             ; Branch if no game on
BPL  RDnoGame           ; (via bit 7 being clear)
LDA  INPT4,X            ; Read Input (Trigger) X.
BPL  Launch             ; unconditional branch -- Launch missile
;
RDnoGame
JSR  MOTORS
JMP  MisKill

MotMis  JSR  MOTORS
JMP  MisAge

MisFly  LDA  AltSnd,X
BEQ  MisBoom
JSR  MOTORS
LDA  #$30
STA  MisLife,X
JMP  MisAge
;
MisBoomLDA  MisLife,X
JSR  BoomSnd
MisAgeLDA  CLOCK            ; Missile aging rate depends on type
AND  #$03
BEQ  MisDec             ; Only do this test 3/4 of the time
BIT  BILLIARD
BVS  MisDSkp            ; branch if Billiard (must bounce before hit)
BIT  PF_PONG
BVC  BMisDec            ; branch if not Pong game (PF_PONG bit 6)
AND  #$01               ; Upshot of this is, in non-billiard Pong
BNE  MisDSkp            ; game, missiles last about twice as long
MisDec  DEC  MisLife,X          ; I'm getting older!
MisDSkpLDA  #$00
BEQ  ResRTS             ; Unconditional -- DO NOT Reset missile to tank
;                       ; (we'd need $02 on to do that) but RTS
;
; Launch a missile
;
LaunchLDA  #$3F
STA  MisLife,X          ; Init MisLife to $3F
SEC
LDA  TankY0,X           ; Copy Y-position... Tank Y-position points
                        ; to top of sprite, but missile is launched
SBC  #$06               ; from its center 6 scanlines down.
STA  MissileY0,X
LDA  DIRECTN,X          ; Copy player bearing to missile.
STA  DIRECTN+2,X
LDA  #$1F
STA  BounceCount,X      ; Init BounceCount to $1F
LDA  #$00
STA  MxPFcount,X        ; Reset MxPFcount
JMP  MisFly             ; Proceed w/missile in flight

; ---------------------------------------------------------------
;
; This routine generates engine or Pong sound as appropriate.
;
MOTORSLDA  AltSnd,X
BEQ  DOMOTOR
; Pong sound.
LDA  #$04
STA  AUDC0,X
LDA  #$07
STA  AUDV0,X
LDA  BounceCount,X
STA  AUDF0,X
RTS
; Engine sound.
DOMOTORLDY  GAMSHP
LDA  SNDV,Y
AND  GameOn             ; Kills sound if no game on by ANDing
STA  AUDV0,X            ; volume value w/$00 no-game value
LDA  SNDC,Y
STA  AUDC0,X
CLC
LDA  #$00
MOPIT0  DEY                     ; This loop sets start value for sound
BMI  MOPIT1             ; pitch based on GAMSHP in Y (tank,
ADC  #$0C               ; biplane, or jet)
BPL  MOPIT0
MOPIT1  ADC  Vtemp,X            ; Use saved velocity to adjust
TAY                     ; sound pitch via SNDP table
TXA
ASL
ADC  SNDP,Y
STA  AUDF0,X
RTS

; ---------------------------------------------------------------
;
; COLISion check
;
; 150 lines of angel-hair spaghetti code
;
; Check to see whether, during all that drawing,
; a missile hit one of the tanks, or a tank hit
; the wall or the other tank, and if so let
; the consequences fall.
;
COLISLDX  #$01                  ; Do first for P1, DEX, P0, etc.
COLnext LDA  CXM0P,X
BPL  COLnoHit           ; No missile collision
BIT  BILLIARD
BVC  COLDET             ; Not Billiard game, go ahead & do it
LDA  BounceCount,X
CMP  #$1F
BEQ  COLnoHit           ; Billiard 1st bounce not satisfied
;
; A touch, a touch!  I do confess.
;
COLDETINC  DIRECTN,X  ; Turn both tanks 22.5 degrees.
INC  DIRECTN+2,X
;
; Increase player's score. A simple INC SCORE,X
; won't do because we're doing it in BCD.
;
SED
LDA  SCORE,X
CLC
ADC  #$01
STA  SCORE,X
CLD
TXA
CLC
ADC  #$FD
STA  StirTimer
;
; Now StirTimer contains loser's ID in bit 0,
; victor's ID in bit 1, and set bits 2-7.
; Bit 1 ID is never used, and just creates a
; slight, unnoticeable difference in stir time.
;
LDA  #$FF
STA  RESMP0             ; Reset both missiles.
STA  RESMP1
LDA  #$00
STA  AUDV0,X            ; Turn off the victor's engine.
STA  MisLife            ; clear MisLife (no missile)
STA  $9A                ; and 9A.
RTS
;
; We didn't just end the game, so we deal with some
; sound and bounce logic
;
COLnoHit
BIT  GAMVAR
BPL  COLTNK             ; Branch if a tank game.
JMP  COLPD              ; Skip this code if NOT a tank game
COLTNKLDA  AltSnd,X
BEQ  COLnoAlt
CMP  #$04               ; See if alt sound has played out
INC  AltSnd,X           ; Increment if it has not
BCC  COLnoAlt
LDA  #$00               ; if played out, reset to 0 "no alt sound"
STA  AltSnd,X
COLnoAlt
LDA  CXM0FB,X           ; Missile collision with playfield?
BMI  COLMPF             ; If true, bounce or obliterate...
LDA  #$00
STA  MxPFcount,X        ; ...else clear MxPFcount
JMP  COLTCK
;
COLMPFBIT  PF_PONG
BVC  COLMISX            ; Branch if not Pong (bit 6 clear)
;
LDA  MxPFcount,X        ; It's Pong, so we bounce
BNE  COLMPFX            ; Branch if collision is already ongoing
INC  AltSnd,X           ; NEW COLLISION, set alt sound flag
DEC  BounceCount,X
LDA  DIRECTN+2,X        ; First try at reflecting
STA  OldMisDir,X        ; Stash current missile heading
EOR  #$FF               ; reverse heading by complement,
STA  DIRECTN+2,X        ; then increment=additive inverse
INC  DIRECTN+2,X        ; same as subtracting from zero
LDA  DIRECTN+2,X        ; check new heading
AND  #$03               ; See if it's moving exactly N,S,E, or W
BNE  COLXY0
INC  DIRECTN+2,X        ; and add 22.5 degrees if so
COLXY0JMP  COLMPFdone
;
; I always wondered how this works.  Stella does not know the
; orientation of the wall that was hit, so this is how it
; reflects:
;
; Immediately after a collision, it tries a vertical reflection,
; jiggering the result so that it won't be exactly vertical or
; exactly horizontal.
;
; If this is the next frame (MxPFcount=$01) that failed, so
; we reverse direction 180 degrees to turn it into a horizontal
; reflection.
;
; On MxPfcount=$02 we take no action, since the missile may need
; the cycle to re-emerge from a wall.
;
```

```
; On MxPFcount=$03 or higher, we retrieve the original heading and
; turn it 180 degrees, assuming a corner reflection.  And we keep
; applying this same bearing until it's out of the #*x@ wall.
;
COLMPFXCMP  #$01              ; branch if
BEQ  Rev180             ; exactly 1 previous collision frame
CMP  #$03               ; branch if
BCC  COLMPFdone         ; less than 3 collision frames
BNE  COLMPFdone         ; or more than three
LDA  OldMisDir,X        ; retrieve pre-bounce missile heading
JMP  Bump180            ; and reverse it 180 degrees
;
; Exactly 1 previous collision:  Do a 180-degree reversal, meaning
; 90 degrees the *other* way from our initial course.
;
Rev180  LDA  DIRECTN+2,X        ; Here to add 180 degrees
Bump180CLC                      ; Here to add A to missile dir
ADC  #$08
STA  DIRECTN+2,X
JMP  COLMPFdone
;
COLMISXLDA  #$01              ; If it's not Pong, we come here and
STA  MisLife,X          ; set the missile's life to 1 to kill it.
;
COLMPFdone                      ; When we're done, increase collision
INC  MxPFcount,X        ; frame count & move on.
;
; Check for tank collisions
;
COLTCK  LDA  CXP0FB,X
BMI  COLTW              ; check if tank collided with a wall.
LDA  CXPPMM             ; check for a tank-tank collision.
BPL  COLTCLR            ; branch if NO tank collisions at all
COLTWLDA  StirTimer          ; See if we are stirring a tank
CMP  #$02
BCC  COLTnk1            ; No, branch & block
JSR  RushTank           ; We are stirring, send it scooting
;
COLTCLRLDA  #$03              ; No tank collision, reset counter
STA  COLcount,X
BNE  COLPD              ; unconditional branch, player done
;
COLTnk1DEC  COLcount,X          ; Tank colliding
BMI  COLBonk            ; COLcount rolled, ignore collision
LDA  Vtemp,X
BEQ  COLPD              ; No boink if velocity=0, player done
BNE  COLreverse         ; else skip INC, needed for elsewhere
;
COLBonkINC  DIRECTN,X          ; Jigger direction 22.5 for disorientation
COLreverse
LDA  DIRECTN,X
CLC
ADC  #$08               ; Add 180 degrees to direction
JSR  BumpTank           ; to bump tank back
;
; COLIS Player Done
;
COLPD   DEX
BMI  COLrts     ;Return if X<0.
JMP  COLnext    ;Else do the other player
COLrts  RTS
;
; Bump the tank in the direction
; the other player's missile is moving
;
RushTank
TXA
EOR  #$01               ; Get OTHER player #
TAY                     ; in Y
LDA  DIRECTN+2,Y        ; OTHER player Missile's Direction
;
; Bump the tank in the direction of a standard
; 22.5-degree bearing code
;
BumpTank
AND  #$0F
TAY
LDA  HDGTBL,Y ;Move
JSR  PhMove   ;Move object in that direction.
LDA  #$00
STA  MVadjA,X
STA  MVadjB,X
STA  FwdTimer,X  ;Stop it dead in its tracks....
LDA  XColor0,X
STA  Color0,X
RTS

; ---------------------------------------------------------------
;
; This was probably a toplevel routine early in development,
; but ended up getting called from GSGRCK. It sets everything
; up to draw the playfield based on the current game selection.
;
InitPFLDX  GAMSHP             ; 0=tank, 1=biplane, 2=jet
LDA  SPRLO,X            ; Set up base pointer to all
STA  SHAPES             ; sprite shapes which will
LDA  SPRHI,X            ; be used in this game.
STA  SHAPES+1
;
LDA  GAMVAR             ; Now set up PF_PONG and playfield type
LSR
LSR
AND  #$03               ; bits 0,1=maze (playfield) type.
TAX                     ; send it to X.
LDA  GAMVAR
BPL  IFgo               ; Branch not plane game, PF_PONG=GAMVAR
AND  #$08               ; Test for clouds
BEQ  IF80               ; Branch if no clouds
LDX  #$03               ; change "maze type" in X to 3 ("clouds")
BPL  IFskip             ; Unconditional skip to next test,
                        ; leaving PF_PONG set to 0.
IF80    LDA  #$80               ; Change PF_PONG to #$80
                        ; (enable playfield, no Pong)
IFgo    STA  PF_PONG            ; store GAMVAR or #$80 in PF_PONG.
IFskip  LDA  GAMVAR             ; Next test..
ASL
ASL                     ; Do this again....
BIT  GAMVAR
BMI  IFnoPlane          ; Branch if a plane game.
STA  WSYNC              ; This MUST be something that dropped
                        ; through the cracks, there is NO reason!
STA  BILLIARD           ; Store GAMVAR*4 in 84 (bit 6 = Billiard Hit)
AND  #$80               ; IF it's a tank game.
IFnoPlane
STA  GUIDED             ; set guided missile flag.
;
; GUIDED is ZERO if a tank game
; it is negative if a guided missile game,
; it is overflowed if a machine gun game.
; (Inapplicable in tank games, hence the
; previous branch trick)
;
LDA  #>PF0_0            ; Store page of first PF map
STA  LORES+1            ; as high order byte
STA  LORES+3            ; for all of these pointers,
STA  LORES+5            ; 'cause that's where it is.
;
; Store the proper offsets for each column of
; playfield from the vectors given
;
LDA  PLFPNT,X
STA  RESP0              ; Reset player 0 while we're at it.
STA  LORES
LDA  PLFPNT+4,X
STA  LORES+2
LDA  PLFPNT+8,X
STA  LORES+4
RTS

; ---------------------------------------------------------------
;
; LoaD STELla
;
; Set the number and size of player sprites, color, and
; disable the joysticks if game is not in play
;
LDSTELLDA  GAMVAR
AND  #$87
BMI  LDmult
;
; If bit 7 is set, we are playing with one or more
; planes.  If not, well, we can only have one tank,
; so...
;
LDA  #$00
LDmultASL
TAX
LDA  WIDTHS,X           ; The TIA's NUSIZ registers make
STA  NUSIZ0             ; it as easy to play with two or
LDA  WIDTHS+1,X         ; three planes as it is for one
STA  NUSIZ1             ; freakin' huge bomber.
LDA  GAMVAR
AND  #$C0
LSR
LSR
LSR
LSR                     ; Our hardware is now in bits 3 and 2.
TAY                     ; Of the Y-register.
;
; Render joysticks immobile if game not in play, and
; select player and field colors according to Y
;
LDA  GameOn             ; Enable joysticks via bit 1
STA  SWCHB              ; of $FF game-on value
EOR  #$FF               ; now $FF=no game, $00=game on
AND  GameTimer          ; Cycle tank colors only when NO
STA  TEMP1              ; game on (attract mode)
LDX  #$FF
LDA  SWCHB
AND  #$08               ; Color/BW switch
BNE  LDcolor            ; Branch if set to Color
LDY  #$10               ; Force B&W colors
LDX  #$0F
LDcolorSTX  TEMP
LDX  #$03               ; We loop 3 times to get 4 values
LDcol0LDA  ColorTbl,Y
EOR  TEMP1              ; Apply color-cycle if no game on
AND  TEMP               ; Apply B&W massage
STA  COLUP0,X           ; Color the real item.
STA  Color0,X           ; Color the virtual item.  This can
                        ; be changd, e.g. invisible tanks
STA  XColor0,X          ; Color the deep virtual item. This
                        ; is used to restore ColorX.
INY
DEX
BPL  LDcol0
RTS
;
; ---------------------------------------------------------------
;
; Zero out zero-page memory starting with ($A3+X) MOD $100,
; through $A2 wrapping around at $100.
;
; Calling with:
; X=$5D will clear $00-$A2
; X=$DD will clear $80-$A2
; X=$DF will clear $82-$A2
; X=$E6 will clear $89-$A2
;
; Returns with zero bit set.
;
ClearMem
LDA  #$00
ClrLoopINX
STA  $A2,X
BNE  ClrLoop  ;Continue until X rolls over.
RTS


;Patterns for numbers
;
NUMBERS.byte $0E ; |    XXX | $F5C5   Leading zero is not drawn
.byte $0A ; |    X X | $F5C6   because it's never used.
.byte $0A ; |    X X | $F5C7
.byte $0A ; |    X X | $F5C8
.byte $0E ; |    XXX | $F5C9

.byte $22 ; |  X   X | $F5CA
.byte $22 ; |  X   X | $F5CB
.byte $22 ; |  X   X | $F5CC
.byte $22 ; |  X   X | $F5CD
.byte $22 ; |  X   X | $F5CE

.byte $EE ; |XXX XXX | $F5CF
.byte $22 ; |  X   X | $F5D0
.byte $EE ; |XXX XXX | $F5D1
.byte $88 ; |X   X   | $F5D2
.byte $EE ; |XXX XXX | $F5D3

.byte $EE ; |XXX XXX | $F5D4
.byte $22 ; |  X   X | $F5D5
.byte $66 ; | XX  XX | $F5D6
.byte $22 ; |  X   X | $F5D7
.byte $EE ; |XXX XXX | $F5D8

.byte $AA ; |X X X X | $F5D9
.byte $AA ; |X X X X | $F5DA
.byte $EE ; |XXX XXX | $F5DB
.byte $22 ; |  X   X | $F5DC
.byte $22 ; |  X   X | $F5DD

.byte $EE ; |XXX XXX | $F5DE
.byte $88 ; |X   X   | $F5DF
.byte $EE ; |XXX XXX | $F5E0
.byte $22 ; |  X   X | $F5E1
.byte $EE ; |XXX XXX | $F5E2

.byte $EE ; |XXX XXX | $F5E3
.byte $88 ; |X   X   | $F5E4
.byte $EE ; |XXX XXX | $F5E5
.byte $AA ; |X X X X | $F5E6
.byte $EE ; |XXX XXX | $F5E7

.byte $EE ; |XXX XXX | $F5E8
.byte $22 ; |  X   X | $F5E9
.byte $22 ; |  X   X | $F5EA
.byte $22 ; |  X   X | $F5EB
.byte $22 ; |  X   X | $F5EC

.byte $EE ; |XXX XXX | $F5ED
.byte $AA ; |X X X X | $F5EE
.byte $EE ; |XXX XXX | $F5EF
.byte $AA ; |X X X X | $F5F0
.byte $EE ; |XXX XXX | $F5F1

.byte $EE ; |XXX XXX | $F5F2
.byte $AA ; |X X X X | $F5F3
.byte $EE ; |XXX XXX | $F5F4
```

```
.byte $22 ; |  X   X | $F5F5
.byte $EE ; |XXX XXX | $F5F6
;
; Horizontal and vertical offsets for movement by orientation.
; Basic table is $10 bytes long (22.5-degree increments), but
; XoffBase is added to it to alter for game options.  High
; nibble is raw HMPx value for horizontal offset, low nibble
; is vertical offset in scan lines.
;
Xoffsets

.BYTE  $F8 ,$F7 ,$F6 ,$06    ;XoffBase=0
.BYTE  $06 ,$06 ,$16 ,$17
.BYTE  $18 ,$19 ,$1A ,$0A
.BYTE  $0A ,$0A ,$FA ,$F9

.BYTE  $F8 ,$F7 ,$F6 ,$F6    ;XoffBase=$10
.BYTE  $06 ,$16 ,$16 ,$17
.BYTE  $18 ,$19 ,$1A ,$1A
.BYTE  $0A ,$FA ,$FA ,$F9

.BYTE  $E8 ,$E6 ,$E4 ,$F4    ;XoffBase=$20
.BYTE  $04 ,$14 ,$24 ,$26    ;normal missiles
.BYTE  $28 ,$2A ,$2C ,$1C
.BYTE  $0C ,$FC ,$EC ,$EA

; This Xoffsets entry is also used directly for "bumping"
; a player after a hit or to back away from playfield collision
;
HDGTBL.BYTE  $C8 ,$C4 ,$C0 ,$E0    ;XoffBase=$30
.BYTE  $00 ,$20 ,$40 ,$44    ;machine guns, "bump"
.BYTE  $48 ,$4C ,$4F ,$2F
.BYTE  $0F ,$EF ,$CF ,$CC
;
; Player velocity momentum adjustments.  Table of two-byte
; entries, indexed by player's desired final velocity.  Even
; locations go to MVadjA to be applied during the first half of
; the FwdTimer cycle, and odd locations goe to MVadjB to be
; applied during the second half.
;
; During each half, the byte is rotated left one bit; if
; the bit which emerges is 1, XoffBase is tweaked by $10
; to adjust the velocity for that frame only.  Since FwdTimer
; goes through 16 cycles or 2 8-bit halves in its course from,
; $F0 to $00, this gives us a bitwise "adjust this frame" flag
; for each frame in the course of FwdTimer's run.  This is
; used to obscure the suddenness of transition from one
; velocity to another.
;
; The adjustment is only done once for each two ON bits
; since the MPace 1 bit is used for the adjustment, and
; MPace is INCed in the same code block that does the
; tweak.  The tweak consists of replacing whatever XoffBase
; the final velocity calls for with $10, an intermediate value.
;
MVtable.BYTE  $00 ,$00
.BYTE  $80 ,$80
.BYTE  $84 ,$20
.BYTE  $88 ,$88
.BYTE  $92 ,$48
.BYTE  $A4 ,$A4
.BYTE  $A9 ,$52
.BYTE  $AA ,$AA
.BYTE  $D5 ,$AA
.BYTE  $DA ,$DA
.BYTE  $DB ,$6D
.BYTE  $EE ,$EE
;
; These are all the sprite shapes.
; The most I suspect any of you will do is
; modify these.  And/or the number shapes.
;
TankShape
        .byte $00 ; |        | $F64F
        .byte $FC ; |XXXXXX  | $F650
        .byte $FC ; |XXXXXX  | $F651
        .byte $38 ; |  XXX   | $F652
        .byte $3F ; |  XXXXXX| $F653
        .byte $38 ; |  XXX   | $F654
        .byte $FC ; |XXXXXX  | $F655
        .byte $FC ; |XXXXXX  | $F656

        .byte $1C ; |   XXX  | $F657
        .byte $78 ; | XXXX   | $F658
        .byte $FB ; |XXXXX XX| $F659
        .byte $7C ; | XXXXX  | $F65A
        .byte $1C ; |   XXX  | $F65B
        .byte $1F ; |   XXXXX| $F65C
        .byte $3E ; |  XXXXX | $F65D
        .byte $18 ; |   XX   | $F65E

        .byte $19 ; |   XX  X| $F65F
        .byte $3A ; |  XXX X | $F660
        .byte $7C ; | XXXXX  | $F661
        .byte $FF ; |XXXXXXXX| $F662
        .byte $DF ; |XX XXXXX| $F663
        .byte $0E ; |    XXX | $F664
        .byte $1C ; |   XXX  | $F665
        .byte $18 ; |   XX   | $F666

        .byte $24 ; |  X  X  | $F667
        .byte $64 ; | XX  X  | $F668
        .byte $79 ; | XXXX  X| $F669
        .byte $FF ; |XXXXXXXX| $F66A
        .byte $FF ; |XXXXXXXX| $F66B
        .byte $4E ; | X  XXX | $F66C
        .byte $0E ; |    XXX | $F66D
        .byte $04 ; |     X  | $F66E

        .byte $08 ; |    X   | $F66F
        .byte $08 ; |    X   | $F670
        .byte $6B ; | XX X XX| $F671
        .byte $7F ; | XXXXXXX| $F672
        .byte $7F ; | XXXXXXX| $F673
        .byte $7F ; | XXXXXXX| $F674
        .byte $63 ; | XX   XX| $F675
        .byte $63 ; | XX   XX| $F676

        .byte $24 ; |  X  X  | $F677
        .byte $26 ; |  X  XX | $F678
        .byte $9E ; |X  XXXX | $F679
        .byte $FF ; |XXXXXXXX| $F67A
        .byte $FF ; |XXXXXXXX| $F67B
        .byte $72 ; | XXX  X | $F67C
        .byte $70 ; | XXX    | $F67D
        .byte $20 ; |  X     | $F67E

        .byte $98 ; |X  XX   | $F67F
        .byte $5C ; | X XXX  | $F680
        .byte $3E ; |  XXXXX | $F681
        .byte $FF ; |XXXXXXXX| $F682
        .byte $FB ; |XXXXX XX| $F683
        .byte $70 ; | XXX    | $F684
        .byte $38 ; |  XXX   | $F685
        .byte $18 ; |   XX   | $F686

        .byte $38 ; |  XXX   | $F687
        .byte $1E ; |   XXXX | $F688
        .byte $DF ; |XX XXXXX| $F689
        .byte $3E ; |  XXXXX | $F68A
        .byte $38 ; |  XXX   | $F68B
        .byte $F8 ; |XXXXX   | $F68C
        .byte $7C ; | XXXXX  | $F68D
        .byte $18 ; |   XX   | $F68E

JetShape
        .byte $60 ; | XX     | $F68F
        .byte $70 ; | XXX    | $F690
        .byte $78 ; | XXXX   | $F691
        .byte $FF ; |XXXXXXXX| $F692
        .byte $78 ; | XXXX   | $F693
        .byte $70 ; | XXX    | $F694
        .byte $60 ; | XX     | $F695
        .byte $00 ; |        | $F696

        .byte $00 ; |        | $F697
        .byte $C1 ; |XX     X| $F698
        .byte $FE ; |XXXXXXX | $F699
        .byte $7C ; | XXXXX  | $F69A
        .byte $78 ; | XXXX   | $F69B
        .byte $30 ; |  XX    | $F69C
        .byte $30 ; |  XX    | $F69D
        .byte $30 ; |  XX    | $F69E

        .byte $00 ; |        | $F69F
        .byte $03 ; |      XX| $F6A0
        .byte $06 ; |     XX | $F6A1
        .byte $FC ; |XXXXXX  | $F6A2
        .byte $FC ; |XXXXXX  | $F6A3
        .byte $3C ; |  XXXX  | $F6A4
        .byte $0C ; |    XX  | $F6A5
        .byte $0C ; |    XX  | $F6A6

        .byte $02 ; |      X | $F6A7
        .byte $04 ; |     X  | $F6A8
        .byte $0C ; |    XX  | $F6A9
        .byte $1C ; |   XXX  | $F6AA
        .byte $FC ; |XXXXXX  | $F6AB
        .byte $FC ; |XXXXXX  | $F6AC
        .byte $1E ; |   XXXX | $F6AD
        .byte $06 ; |     XX | $F6AE

        .byte $10 ; |   X    | $F6AF
        .byte $10 ; |   X    | $F6B0
        .byte $10 ; |   X    | $F6B1
        .byte $38 ; |  XXX   | $F6B2
        .byte $7C ; | XXXXX  | $F6B3
        .byte $FE ; |XXXXXXX | $F6B4
        .byte $FE ; |XXXXXXX | $F6B5
        .byte $10 ; |   X    | $F6B6

        .byte $40 ; | X      | $F6B7
        .byte $20 ; |  X     | $F6B8
        .byte $30 ; |  XX    | $F6B9
        .byte $38 ; |  XXX   | $F6BA
        .byte $3F ; |  XXXXXX| $F6BB
        .byte $3F ; |  XXXXXX| $F6BC
        .byte $78 ; | XXXX   | $F6BD
        .byte $60 ; | XX     | $F6BE

        .byte $40 ; | X      | $F6BF
        .byte $60 ; | XX     | $F6C0
        .byte $3F ; |  XXXXXX| $F6C1
        .byte $1F ; |   XXXXX| $F6C2
        .byte $1E ; |   XXXX | $F6C3
        .byte $1E ; |   XXXX | $F6C4
        .byte $18 ; |   XX   | $F6C5
        .byte $18 ; |   XX   | $F6C6

        .byte $00 ; |        | $F6C7
        .byte $83 ; |X     XX| $F6C8
        .byte $7F ; | XXXXXXX| $F6C9
        .byte $3E ; |  XXXXX | $F6CA
        .byte $1E ; |   XXXX | $F6CB
        .byte $0C ; |    XX  | $F6CC
        .byte $0C ; |    XX  | $F6CD
        .byte $0C ; |    XX  | $F6CE
PlaneShape
        .byte $00 ; |        | $F6CF
        .byte $8E ; |X   XXX | $F6D0
        .byte $84 ; |X    X  | $F6D1
        .byte $FF ; |XXXXXXXX| $F6D2
        .byte $FF ; |XXXXXXXX| $F6D3
        .byte $04 ; |     X  | $F6D4
        .byte $0E ; |    XXX | $F6D5
        .byte $00 ; |        | $F6D6

        .byte $00 ; |        | $F6D7
        .byte $0E ; |    XXX | $F6D8
        .byte $04 ; |     X  | $F6D9
        .byte $8F ; |X   XXXX| $F6DA
        .byte $7F ; | XXXXXXX| $F6DB
        .byte $72 ; | XXX  X | $F6DC
        .byte $07 ; |     XXX| $F6DD
        .byte $00 ; |        | $F6DE

        .byte $10 ; |   X    | $F6DF
        .byte $36 ; |  XX XX | $F6E0
        .byte $2E ; |  X XXX | $F6E1
        .byte $0C ; |    XX  | $F6E2
        .byte $1F ; |   XXXXX| $F6E3
        .byte $B2 ; |X XX  X | $F6E4
        .byte $E0 ; |XXX     | $F6E5
        .byte $40 ; | X      | $F6E6

        .byte $24 ; |  X  X  | $F6E7
        .byte $2C ; |  X XX  | $F6E8
        .byte $5D ; | X XXX X| $F6E9
        .byte $1A ; |   XX X | $F6EA
        .byte $1A ; |   XX X | $F6EB
        .byte $30 ; |  XX    | $F6EC
        .byte $F0 ; |XXXX    | $F6ED
        .byte $60 ; | XX     | $F6EE

        .byte $18 ; |   XX   | $F6EF
        .byte $5A ; | X XX X | $F6F0
        .byte $7E ; | XXXXXX | $F6F1
        .byte $5A ; | X XX X | $F6F2
        .byte $18 ; |   XX   | $F6F3
        .byte $18 ; |   XX   | $F6F4
        .byte $18 ; |   XX   | $F6F5
        .byte $78 ; | XXXX   | $F6F6

        .byte $34 ; |  XX X  | $F6F7
        .byte $36 ; |  XX XX | $F6F8
        .byte $5A ; | X XX X | $F6F9
        .byte $78 ; | XXXX   | $F6FA
        .byte $2C ; |  X XX  | $F6FB
        .byte $0C ; |    XX  | $F6FC
        .byte $06 ; |     XX | $F6FD
        .byte $0C ; |    XX  | $F6FE

        .byte $08 ; |    X   | $F6FF
        .byte $6C ; | XX XX  | $F700
        .byte $70 ; | XXX    | $F701
        .byte $B8 ; |X XXX   | $F702
        .byte $DC ; |XX XXX  | $F703
        .byte $4E ; | X  XXX | $F704
        .byte $07 ; |     XXX| $F705
        .byte $06 ; |     XX | $F706

        .byte $38 ; |  XXX   | $F707
        .byte $10 ; |   X    | $F708
        .byte $F0 ; |XXXX    | $F709
        .byte $7C ; | XXXXX  | $F70A
        .byte $4F ; | X  XXXX| $F70B
        .byte $E3 ; |XXX   XX| $F70C
        .byte $02 ; |      X | $F70D
```

```
        .byte $00 ; |        | $F70E
;
; These are sub-pointers, used to set up the
; two-dimensional array at CTRLTBL.
;
CtrlBase .BYTE  $00 ,$0B ,$16

;
; Two-dimensional array, 12x3.
;
; This array specifies what the joystick does
; in each game.  Looking at it now the format looks
; like this:
;
;   Low nybble = Amount to rotate object (signed)
;                   $00 = Not at all
;                   $01 = Clockwise         (+1)
;                   $0F = Counter-clockwise (-1)
;   High nybble = Speed to move object (unsigned)
;                   $00 = Not moving
;                   $F0 = Warp speed
;
; Observe the $FF's.  Notice how indexing out of bounds with impossible
; joystick movements will cause strange behavior.
;
; Tank movement
;                   UP  DOWN (No reverse)
CTRLTBL .BYTE  $00 ,$10 ,$00 ,$FF
.BYTE  $01 ,$11 ,$01 ,$FF ;LEFT
.BYTE  $0F ,$1F ,$0F      ;RIGHT
;
; Biplane movement (This is why controls are sideways)
;                   UP  DOWN
.BYTE  $50 ,$5F ,$51 ,$FF ;
.BYTE  $30 ,$3F ,$31 ,$FF ;LEFT
.BYTE  $70 ,$7F ,$71      ;RIGHT
;
; Jet fighter movement
;                   UP  DOWN
.BYTE  $90 ,$B0 ,$70 ,$FF ;
.BYTE  $91 ,$B1 ,$71 ,$FF ;LEFT
.BYTE  $9F ,$BF ,$7F      ;RIGHT
;
;
; Sound information for different game types.
; Different tools of destruction make different
; sound.
;
; There is some more data below which looks to
; be other information: different machines at
; different speeds.  The pitch table is 3D,
; having 12-entry records for each GAMSHP.
;
;          Tanks Biplane, Jet Fighter
SNDV.BYTE  $08 ,$02 ,$02 ; sound volumes
SNDC.BYTE  $02 ,$03 ,$08 ; sound types

SNDP.BYTE  $1D ,$05 ,$00 ; sound pitches indexed by velocity
.BYTE  $00 ,$00 ,$00 ; for TANKS
.BYTE  $00 ,$00 ,$00
.BYTE  $00 ,$00 ,$00

.BYTE  $00 ,$00 ,$1D ; for BIPLANES
.BYTE  $1D ,$16 ,$16
.BYTE  $0F ,$0F ,$00
.BYTE  $00 ,$00 ,$00

.BYTE  $00 ,$00 ,$00 ; for JETS
.BYTE  $00 ,$00 ,$12
.BYTE  $10 ,$10 ,$0C
.BYTE  $0C ,$07 ,$07
;
; Player widths for various plane games.
; Through the miracle of the Atari 2600's NUSIZ
; register, the difference between a 1 vs. 1 game
; and a Bomber vs. 3 game is contained in just
; two bytes.
;
WIDTHS.BYTE  $00 ,$00  ;1 vs. 1
.BYTE  $01 ,$01  ;2 vs. 2
.BYTE  $00 ,$03  ;1 vs. 3
.BYTE  $27 ,$03  ;Bomber vs. 3

; Table of color combinations.  Each 4 byte entry specifies
; Player 0, Player1, Playfield, and Background colors.
; (By a not-so-odd coincidence, these 4 color registers are
; addressed consecutively in the same order in the TIA.)
; Table is indexed by the high 2 bits of GAMVAR << 2, or
; forced to +$10 if B&W switch selected.
;
ColorTbl
byte $EA ,$3C ,$82 ,$44      ; 00 = Regular Tanks
        .byte $32 ,$2C ,$8A ,$DA      ; 01 = Tank Pong
        .byte $80 ,$9C ,$DA ,$3A      ; 10 = Jets
        .byte $64 ,$A8 ,$DA ,$4A      ; 11 = Biplanes
        .byte $08 ,$04 ,$00 ,$0E      ; special B&W

PF0_0   .byte $F0 ; |XXXX    | $F779
        .byte $10 ; |    X    | $F77A
        .byte $10 ; |    X    | $F77B
        .byte $10 ; |    X    | $F77C
        .byte $10 ; |    X    | $F77D
        .byte $10 ; |    X    | $F77E
        .byte $10 ; |    X    | $F77F
        .byte $10 ; |    X    | $F780
        .byte $10 ; |    X    | $F781
        .byte $10 ; |    X    | $F782
        .byte $10 ; |    X    | $F783
        .byte $10 ; |    X    | $F784
PF1_0   .byte $FF ; |XXXXXXXX| $F785
        .byte $00 ; |        | $F786
        .byte $00 ; |        | $F787
        .byte $00 ; |        | $F788
        .byte $38 ; |  XXX   | $F789
        .byte $00 ; |        | $F78A
        .byte $00 ; |        | $F78B
        .byte $00 ; |        | $F78C
        .byte $60 ; | XX     | $F78D
        .byte $20 ; |  X     | $F78E
        .byte $20 ; |  X     | $F78F
        .byte $23 ; |  X   XX| $F790
PF2_0   .byte $FF ; |XXXXXXXX| $F791
        .byte $80 ; |X       | $F792
        .byte $80 ; |X       | $F793
        .byte $00 ; |        | $F794
        .byte $00 ; |        | $F795
        .byte $00 ; |        | $F796
        .byte $1C ; |   XXX  | $F797
        .byte $04 ; |     X  | $F798
        .byte $00 ; |        | $F799
        .byte $00 ; |        | $F79A
        .byte $00 ; |        | $F79B
        .byte $00 ; |        | $F79C
PF1_1   .byte $FF ; |XXXXXXXX| $F79D
PF0_3   .byte $00 ; |        | $F79E
        .byte $00 ; |        | $F79F
        .byte $00 ; |        | $F7A0
PF1_3   .byte $00 ; |        | $F7A1
        .byte $00 ; |        | $F7A2
        .byte $00 ; |        | $F7A3
        .byte $00 ; |        | $F7A4
        .byte $00 ; |        | $F7A5
        .byte $00 ; |        | $F7A6
        .byte $00 ; |        | $F7A7
        .byte $00 ; |        | $F7A8
        .byte $00 ; |        | $F7A9
        .byte $07 ; |     XXX| $F7AA
        .byte $1F ; |   XXXXX| $F7AB
        .byte $3F ; |  XXXXXX| $F7AC
        .byte $7F ; | XXXXXXX| $F7AD
PF1_2   .byte $FF ; |XXXXXXXX| $F7AE
        .byte $00 ; |        | $F7AF
        .byte $00 ; |        | $F7B0
        .byte $00 ; |        | $F7B1
        .byte $00 ; |        | $F7B2
        .byte $00 ; |        | $F7B3
        .byte $00 ; |        | $F7B4
        .byte $00 ; |        | $F7B5
        .byte $00 ; |        | $F7B6
        .byte $60 ; | XX     | $F7B7
        .byte $20 ; |  X     | $F7B8
        .byte $21 ; |  X    X| $F7B9
PF2_2   .byte $FF ; |XXXXXXXX| $F7BA
        .byte $00 ; |        | $F7BB
        .byte $00 ; |        | $F7BC
        .byte $00 ; |        | $F7BD
        .byte $80 ; |X       | $F7BE
        .byte $80 ; |X       | $F7BF
        .byte $80 ; |X       | $F7C0
        .byte $80 ; |X       | $F7C1
        .byte $00 ; |        | $F7C2
        .byte $00 ; |        | $F7C3
        .byte $00 ; |        | $F7C4
        .byte $07 ; |     XXX| $F7C5

; Addresses for Sprite Graphics

SPRLO   .BYTE  #PlaneShape, #TankShape, #>PlaneShape, #>JetShape


; Playfield address data.  Kernal timing requires that
; these addresses point 4 bytes before the real start
; of data.
;
;       Complex   ,    None
;       Simple    ,   Clouds
PLFPNT.BYTE  #<(PF0_0-4) ,#<(PF0_0-4)
.BYTE  #<(PF0_0-4) ,#<(PF0_3-4)   ;PF0
.BYTE  #<(PF1_0-4) ,#<(PF1_1-4)
.BYTE  #<(PF1_2-4) ,#<(PF1_3-4)   ;PF1
.BYTE  #<(PF2_0-4) ,#<(PF1_1-4)
.BYTE  #<(PF2_2-4) ,#<(PF1_3-4)   ;PF2

; Game features, indexed by game number-1.
;
; bits
;   1,0:      TANKS           PLANES
;       X0 = Normal
;       X1 = Invisible
;       00 =                  1 vs. 1
;       01 =                  2 vs. 2
;       10 =                  3 vs. 1
;       11 =                  3 vs. Giant
;   3,2: 01 = No maze
;        10 = Simple maze
;        00 = Complex maze
;        1X =                 Clouds
;        0X =                 No clouds
;     4:  0 = Direct Hit      Normal Gun
;         1 = Billiard Hit    Machine Gun
;     5:  0 =     Straight Missiles
;         1 =     Guided Missiles
;     6:  0 = Tanks           Jets
;         1 = Tank Pong       Biplanes
;     7:  0 =     Tank Game
;         1 =     Plane Game

;
VARMAP.BYTE  $24 ;Game 1:  0010 0100  TANK
.BYTE  $28 ;Game 2:  0010 1000
.BYTE  $08 ;Game 3:  0000 1000
.BYTE  $20 ;Game 4:  0010 0000
.BYTE  $00 ;Game 5:  0000 0000
.BYTE  $48 ;Game 6:  0100 1000  TANK PONG
.BYTE  $40 ;Game 7:  0100 0000
.BYTE  $54 ;Game 8:  0101 0100
.BYTE  $58 ;Game 9:  0101 1000
.BYTE  $25 ;Game 10: 0010 0101  INVISIBLE TANK
.BYTE  $29 ;Game 11: 0010 1001
.BYTE  $49 ;Game 12: 0100 1001  INVISIBLE TANK-PONG
.BYTE  $55 ;Game 13: 0101 0101
.BYTE  $59 ;Game 14: 0101 1001
.BYTE  $A8 ;Game 15: 1010 1000  BIPLANE
.BYTE  $88 ;Game 16: 1000 1000
.BYTE  $98 ;Game 17: 1001 1000
.BYTE  $90 ;Game 18: 1001 0000
.BYTE  $A1 ;Game 19: 1010 0001
.BYTE  $83 ;Game 20: 1000 0011
.BYTE  $E8 ;Game 21: 1110 1000  JET FIGHTER
.BYTE  $C8 ;Game 22: 1100 1000
.BYTE  $E0 ;Game 23: 1110 0000
.BYTE  $C0 ;Game 24: 1100 0000
.BYTE  $E9 ;Game 25: 1110 1001
.BYTE  $E2 ;Game 26: 1110 0010
.BYTE  $C1 ;Game 27: 1100 0001
;
; $FF to signify end of game variations.
;
.BYTE  $FF

; If you were changing this to a 4K cart, you'd
; want to change this ORG to $FFFC.  You might also
; want to move AudPitch out of the interrupt vector...
;
ORG $F7FC
.word $f000             ; Reset IRQ
;
AudPitch
.BYTE  $0F, $11         ; Motor sound pitch table by player
```