```
;=================================================================
==================================
;
PLAYER.ASM  - PLAYER LOGIC
;
Peter 'Sig' Hewett - Retroromicon 2017
;=================================================================
==================================
;
; Handling player and player control logic
;-----------------------------------------------------------------
----------------------------------
;-----------------------------------------------------------------
----------------------------------
;
DEFINES
;-----------------------------------------------------------------
----------------------------------
MOB_ISACTIVE = 0

;-----------------------------------------------------------------
----------------------------------
;
PLAYER SETUP
;-----------------------------------------------------------------
----------------------------------
#region "Player Setup"
PlayerSetup

        ;-------------------------------------------------------
-----------------------
        ; PLAYER has a strange setup as it's ALWAYS going to be
using sprites 0 and 1
        ; As well as always being 'active' (used)
        ;-------------------------------------------------------
-----------------------

        lda #%00000011                          ; Turn on
multicolor for sprites 0 and 1
        sta VIC_SPRITE_MULTICOLOR               ; also turn all
others to single color

        lda #COLOR_BROWN
        sta VIC_SPRITE_MULTICOLOR_1             ; Set sprite
shared multicolor 1 to brown
        lda #COLOR_LTRED
```

```
        sta VIC_SPRITE_MULTICOLOR_2             ; set sprite
shared multicolor 2 to 'pink'

        lda #COLOR_YELLOW
        sta VIC_SPRITE_COLOR                    ; set sprite 0
color to yellow
        lda #COLOR_ORANGE
        sta VIC_SPRITE_COLOR + 1                ; set sprite 1
color to orange (bkground sprite)


        ;-------------------------------------------------------
-----------------------
        ; We now use a system that tracks the sprite position in
character coords on
        ; the screen, so to avoid costly calculations every
frame, we set the sprite
        ; to a character border intially and track all movement
from there. That way
        ; we need only do this set of calculations once in the
lifetime of the Player.
        ;
        ; To initally place the sprite, we use 'SpriteToCharPos'
        ;-------------------------------------------------------
-----------------------

        lda #19
        sta PARAM1                          ; Char X pos = 19
        lda #11
        sta PARAM2                          ; Char Y pos = 10

        ldx #0                              ; Sprite number 0
        jsr SpriteToCharPos                 ; Set sprite and store
coords

        ldx #1                              ; Sprite number 1
        jsr SpriteToCharPos                 ; Set sprite and store
coords

        ;-------------------------------------------------------
--------------------
        ; Set sprite images.  The sprites from the MLP Spelunker
demo used 2 sprites
        ; overlapped so they could use an extra color.  So our
main player sprite
        ; uses 2 sprites (0 and 1).  The first walking frame
image 1, and it's
```

```
        ; background sprite is image 8.  We use the
SetSpriteImage subroutine as it
        ; will update the pointers for both Screen1 and Screen2
for us.
        ;-------------------------------------------------------
--------------------

        lda #PLAYER_STATE_IDLE          ; Set initial state
(idle)
        jsr ChangePlayerState

        ;-------------------------------- This will be relevant
later
;        lda #3                         ; Trim for better
collisions
        lda #0
        sta SPRITE_DELTA_TRIM_X

        ;-------------------------------- Set Player Mob active
        lda #1
        sta SPRITE_IS_ACTIVE            ; Set sprite 0 to active
        sta SPRITE_IS_ACTIVE + 1        ; Set sprite 1 to active

        lda #0                          ; reset the player
fallcount
        sta PLAYER_FALLCOUNT
        rts

#endregion

;=======================================================================
====================================
;
UPDATE PLAYER
;-------------------------------------------------------------------
-----------------------------------
; Update the player. Joystick controls are updated via interrupt
so we read the values from JOY_X
; and JOY_Y
;-------------------------------------------------------------------
-----------------------------------
#region "Update Player"

PLAYER_RIGHT_CAP = $1c                  ; Sprite movement
caps - at this point we don't
PLAYER_LEFT_CAP = $09                   ; Move the sprite,
we scroll the screen
```

```
PLAYER_UP_CAP = $04
PLAYER_DOWN_CAP = $0F

UpdatePlayer
                                                ; Only update the
player if it's active
        lda SPRITE_IS_ACTIVE                    ; check against
sprite #0 - is it active?
        bne @update
        rts
@update
        ldx #0
        jsr AnimateSprite                           ; Animate sprite
        jsr UpdatePlayerState                       ; Update player
by state

        rts

#endregion

;=================================================================
==================================
;
JOYSTICK / PLAYER MOVE
;=================================================================
==================================
; The old system of joystick movement was going to become very
unweildy very fast and not be very
; good for expanding what the player can do. I'm trying a new
system where the routines are broken
; down and input is checked in individual states for what the
player can do at any given time.
; The movement routines will then be broken down and called as
needed by the states.
; Since the old system didn't actually read the joystick or
scroll the screen (it read / set
; variables by routines that do) - this SHOULD be fairly
workable.
;-----------------------------------------------------------------
----------------------------------

;=================================================================
==================================
;
JOYSTICK READY
;-----------------------------------------------------------------
----------------------------------
```

```
; There are times atm when we have to ignore joystick input so
the scrolling can 'catch up' after
; movement stops. Usually for a couple of frames.
;
; Returns A :  0 = ready   1 = not ready
;
; Modifies A
;-----------------------------------------------------------------
-----------------------------------
#region "JoystickReady"
JoystickReady

        lda SCROLL_MOVING               ; if moving is 'stopped'
we can test joystick
        beq @joyready
                                        ; if it's moving but
direction is stopped, we're 'fixing'
        lda SCROLL_DIRECTION
        bne @joyready

        lda #1                          ; Send code for joystick
NOT ready for input
        rts

@joyready
        lda #SCROLL_STOP                ; reset scroll direction
- if it needs to scroll
        sta SCROLL_DIRECTION            ; it will be updated

        lda #0                          ; send code for joystick
ready
        rts

#endregion

;=================================================================
====================================
;
MOVE PLAYER RIGHT
;-----------------------------------------------------------------
-----------------------------------
; Move the player one pixel to the right if possible, taking
into account scrolling, map limits
; and collision detection against the screen
;
; Returns A: any blocking or special character to the right, or
0 if clear
```

```
;
;-------------------------------------------------------------------
-----------------------------------
#region "MovePlayerRight"
MovePlayerRight
        lda #0
        sta SCROLL_FIX_SKIP
        ;---------------------------------------- CHECK RIGHT
MOVEMENT CAP
        clc                             ; clear carry flag
because I'm paranoid
        lda SPRITE_CHAR_POS_X           ; load the sprite char X
position
        cmp #PLAYER_RIGHT_CAP           ; check against the
right edge of the screen
        bcc @rightMove                  ; if X char pos < cap -
move the sprite, else scroll

                                        ; Check against map edge
        lda MAP_X_POS                   ; load the current MAP X
Position
        cmp #54                         ; the map is 64 tiles
wide, the screen is 10 tiles wide
        bne @scrollRight
        lda MAP_X_DELTA                 ; each tile is 4
characters wide (0-3)
        cmp #1                          ; if we hit this limit
we don't scroll (or move)
        bne @scrollRight
                                        ;at this point we will
revert to move
        lda #1
        sta SCROLL_FIX_SKIP
        jmp @rightMove
        rts
        ;---------------------------------------- SCROLL RIGHT
                                        ; Pre-scroll check
@scrollRight
        ldx #0
        jsr CheckMoveRight              ; Collision check
against characters
        beq @scroll                     ; TODO - return the
collision code here
        rts
                                        ; Setup for the scroll
@scroll
```

```
        lda #SCROLL_RIGHT               ; Set the direction for
scroll and post scroll checks
        sta SCROLL_DIRECTION
        sta SCROLL_MOVING
        lda #0                          ; load 'clear code'
        rts                             ; TODO - ensure
collision code is returned

        ;-------------------------------------- MOVE SPRITE
RIGHT
@rightMove
        ldx #0
        jsr CheckMoveRight              ; Check ahead for
character collision
        bne @rightDone

        jsr MoveSpriteRight             ; Move sprites one pixel
right
        ldx #1
        jsr MoveSpriteRight
        lda #0                          ; move code 'clear'
@rightDone
        rts

#endregion

;=================================================================
==================================
;
MOVE PLAYER LEFT
;-----------------------------------------------------------------
----------------------------------
; Move the player one pixel to the left if possible, taking into
account scrolling, map limits
; and collision detection against the screen
;
; Returns A: any blocking or special character to the right, or
0 if clear
;-----------------------------------------------------------------
----------------------------------
#region "Move Player Left"
MovePlayerLeft
        lda #0                          ; Make sure scroll 'fix'
is on
        sta SCROLL_FIX_SKIP
        ;-------------------------------------- CHECK MOVEMENT
CAP ($07)
```

```
        lda SPRITE_CHAR_POS_X           ; Check for left side
movement cap
        cmp #PLAYER_LEFT_CAP
        bcs @leftMove                   ; if below cap, we move
the sprite
                                        ; Otherwise we prepare
to scroll

                                        ; Check for edge of map
for scrolling
        lda MAP_X_POS                   ; Check for map pos X =
0
        bne @scrollLeft
        lda MAP_X_DELTA                 ; check for map delta =
0
        bne @scrollLeft
                                        ; We're at the maps left
edge
                                        ; So we revert to sprite
movement once more
        lda #1
        sta SCROLL_FIX_SKIP
        lda SPRITE_POS_X,x              ; Check for sprite pos >
0 (not sprite char pos)
        bpl @leftMove                   ; so we could walk to
the edge of screen
        rts

@scrollLeft
        ;-------------------------------------- SCROLL SCREEN
FOR LEFT MOVE
        ldx #0
        jsr CheckMoveLeft               ; check for character
collision to the left
        beq @scroll
        rts                             ; TODO - return
collision code

@scroll
        lda #SCROLL_LEFT
        sta SCROLL_DIRECTION
        sta SCROLL_MOVING

        lda #0                          ; return 'clear code'
                                        ; TODO - return clear
collision code
        rts
```

```
        ;-------------------------------------- MOVE THE
PLAYER LEFT ONE PIXEL
@leftMove
        ldx #0
        jsr CheckMoveLeft               ; check for collisions
with characters
        bne @leftDone                   ; TODO return collision
code

        ldx #0
        jsr MoveSpriteLeft
        ldx #1
        jsr MoveSpriteLeft
        lda #0                          ; move code 'clear'

@leftDone
        rts

#endregion
;================================================================
==================================
;
MOVE PLAYER DOWN
;----------------------------------------------------------------
----------------------------------
; Move the player one pixel down if possible, taking into
account scrolling, map limits
; and collision detection against the screen
;
; Returns A: any blocking or special character below, or 0 if
clear
;
; Modifies X
;----------------------------------------------------------------
----------------------------------
#region "Move Player Down"
MovePlayerDown
        clc
        jmp @downMove
        lda SPRITE_CHAR_POS_Y
        cmp #PLAYER_DOWN_CAP
        bcc @downMove

        lda MAP_Y_POS
        cmp #$1B
        bne @downScroll
        lda MAP_Y_DELTA
```

```
        cmp #02
        bcc @downScroll
        rts

@downScroll

        ldx #0
        jsr CheckMoveDown
        beq @scroll
        rts                             ; return with contents
of collison routine

@scroll
        lda #SCROLL_DOWN
        sta SCROLL_DIRECTION
        sta SCROLL_MOVING
        lda #0                          ; return a clear
collision code
        rts

@downMove
        ;ldx #0
        ;jsr CheckMoveDown
        ;bne @downDone                   ; retun with contents
of collision code
        jsr MoveSpriteDown
        ldx #1
        jsr MoveSpriteDown
        lda #0                          ; return with clear code
@downDone
        rts

#endregion

;=================================================================
====================================
;
MOVE PLAYER UP
;-----------------------------------------------------------------
-----------------------------------
; Move the player one pixel up if possible, taking into account
scrolling, map limits
; and collision detection against the screen
;
; Returns A: any blocking or special character below, or 0 if
clear
```

```
;-------------------------------------------------------------
----------------------------------
#region "MovePlayerUp"
MovePlayerUp
        sec
        lda SPRITE_CHAR_POS_Y
        cmp #PLAYER_UP_CAP
        bcs @upMove

        lda MAP_Y_POS
        bne @upScroll
        clc
        lda MAP_Y_DELTA
        cmp #1
        bcs @upScroll
        rts

@upScroll
        ldx #0
        jsr CheckMoveUp
        beq @scroll
        rts

@scroll
        lda #SCROLL_UP
        sta SCROLL_DIRECTION
        sta SCROLL_MOVING
        rts

@upMove
        ldx #0
        jsr CheckMoveUp
        bne @upDone

        jsr MoveSpriteUp
        ldx #1
        jsr MoveSpriteUp
@upDone
        rts

#endregion
;=============================================================
====================================
;
APPLY GRAVITY
;=============================================================
====================================
```

```
; Apply Gravity to the player - this system will be totally
rewritten at some point to apply
; a proper gravity to a player or any other sprite.. but for now
it's just super basic
;
; A returns 0 if we moved down and a collision code if we didn't
;-------------------------------------------------------------
-----------------------------------
#region "Apply Gravity"
ApplyGravity

        ;jsr MovePlayerDown
@done
        rts
#endregion
;-------------------------------------------------------------
-----------------------------------
;=============================================================
===================================
;
PLAYER STATES
;=============================================================
===================================
; Player states are incremented by 2 as they are indexes to look
up the address of the state
; code on the PLAYER_STATE_JUMPTABLE.  An address is 2 bytes (1
word) egro the index must increase
; by 2 bytes.
;-------------------------------------------------------------
-----------------------------------
PLAYER_STATE_IDLE       = 0     ; standing still - awaiting
input
PLAYER_STATE_WALK_R     = 2     ; Walking right
PLAYER_STATE_WALK_L     = 4     ; Walking left
PLAYER_STATE_FALL       = 6     ; Falling
PLAYER_STATE_STAIRS_R   = 8     ; stairs on the right
PLAYER_STATE_STAIRS_L   = 10    ; stairs on the left
PLAYER_STATE_ROPE       = 12    ; climb rope

PLAYER_SUBSTATE_ENTER   = 0     ; we have just entered this
state
PLAYER_SUBSTATE_RUNNING = 1     ; This state is running normally

;-------------------------------------------------------------
-----------------------------------
;
PLAYER STATE JUMPTABLE
```

```
;-------------------------------------------------------------------
-----------------------------------

PLAYER_STATE_JUMPTABLE
        word PlayerStateIdle
        word PlayerStateWalkR
        word PlayerStateWalkL
        word PlayerStateFall
        word PlayerStateStairsR
        word PlayerStateStairsL
        word PlayerStateRope


;-------------------------------------------------------------------
-----------------------------------
;===================================================================
===================================
;
CHANGE PLAYER STATE
;-------------------------------------------------------------------
-----------------------------------
; Change a players state
;
; A = state to change to
;
; Modifies A,X,ZEROPAGE_POINTER_1
;-------------------------------------------------------------------
-----------------------------------
#region "PlayerChangeState"
ChangePlayerState
        tax                                             ;
transfer A to X
        stx PLAYER_STATE                                ; store
the new player state
        lda #PLAYER_SUBSTATE_ENTER                      ; Set
substate to ENTER
        sta PLAYER_SUBSTATE

        lda #1
        sta SPRITE_ANIM_PLAY

        lda PLAYER_STATE_JUMPTABLE,x                    ; lookup
state to change to
        sta ZEROPAGE_POINTER_1                          ; and
store it in ZEROPAGE_POINTER_1

        lda PLAYER_STATE_JUMPTABLE + 1,x
```

```
        sta ZEROPAGE_POINTER_1 + 1

        jmp (ZEROPAGE_POINTER_1)                        ; jump
to state (to setup)
                                                        ; NOTE:
This is NOT a jsr.
                                                        ; The
state will act as an extension of
                                                        ; this
routine then return.
        rts
#endregion
;=================================================================
===================================
;
UPDATE PLAYER STATE
;-----------------------------------------------------------------
-----------------------------------
; Update the player based on their state
;-----------------------------------------------------------------
-----------------------------------
#region "UpdatePlayerState"
UpdatePlayerState
        ldx PLAYER_STATE                        ; Load player
state
        lda PLAYER_STATE_JUMPTABLE,x            ; fetch the
state address from the jump table
        sta ZEROPAGE_POINTER_1                  ; store it in
ZEROPAGE_POINTER_1
        lda PLAYER_STATE_JUMPTABLE +1,x
        sta ZEROPAGE_POINTER_1 + 1
        jmp (ZEROPAGE_POINTER_1)                ; jump to the
right state (note - NOT a jsr)

        rts
#endregion

;=================================================================
===================================
;
STATE IDLE
;-----------------------------------------------------------------
-----------------------------------
; The player is standing still and waiting input.
; Possible optimizations we are doublechecking CheckBlockUnder
and CheckDown, we can check once
; and store those in a temp variable and look them up if needed.
```

```
;-------------------------------------------------------------
-----------------------------------

#region "Player State Idle"
PlayerStateIdle
        lda PLAYER_SUBSTATE                     ; Check for
first entry to state
        bne @running

        ldx #0                                  ; load sprite
number (0) in X
        lda #<ANIM_PLAYER_IDLE                  ; load animation
list in ZEROPAGE_POINTER_1
        sta ZEROPAGE_POINTER_1
        lda #>ANIM_PLAYER_IDLE
        sta ZEROPAGE_POINTER_1 + 1

        jsr InitSpriteAnim                      ; setup the
animation for Idle
        lda PLAYER_SUBSTATE_RUNNING             ; set the
substate to Running
        sta PLAYER_SUBSTATE
        rts                                     ; wait till next
frame to start

@running
        ;-------------------------------------------------------
----- JOYSTICK INPUT
        lda #0                                  ; clear the idle
variable
        sta IDLE_VAR

        jsr JoystickReady
        beq @input
        rts                                     ; not ready for
input, we return

@input                                          ; process valid
joystick input

;        jsr CheckBlockUnder                     ; Check for
standing on a special block
;        cmp #COLL_ROPE
;        beq @horizcheck

        jsr ApplyGravity                        ; Apply gravity
and test for ground
```

```
@horizCheck
        lda JOY_X                               ; horizontal
movement
        beq @vertCheck                          ; check zero -
ho horizontal input
        bmi @left                               ; negative =
left
        ;------------------------------------------ JOYSTICK
RIGHT
@right
        lda #PLAYER_STATE_WALK_R                ; go to walk
state right
        jmp ChangePlayerState

        ;------------------------------------------ JOYSTICK
LEFT
@left
        lda #PLAYER_STATE_WALK_L                ; go to walk
state left
        jmp ChangePlayerState

@vertCheck
        lda JOY_Y                               ; check vertical
joystick input
        beq @end                                ; zero means no
input
        bmi @up                                 ; negative means
up
        bpl @down                               ; already
checked for 0 - so this is positive
        rts

        ;------------------------------------------ JOYSTICK
UP
@up
        ldx #0
        ;jsr CheckBlockUnder
        cmp #COLL_ROPE                          ; Check for rope
under player
        bne @end
        lda #PLAYER_STATE_ROPE                  ; change to
climb rope state
        jmp ChangePlayerState

        ;------------------------------------------ JOYSTICK
DOWN
@down
```

```
        ldx #0                          ; if we are on a rope,
can we move down?
        ;jsr CheckBlockUnder             ; first check we are on
a rope
        cmp #COLL_ROPE
        bne @noRope

        jsr CheckMoveDown               ; If we are at the end,
there will be solid ground under us
        beq @goRopeClimb                ; No blocking and on
rope? We change to climbing

                                        ; Otherwise we have more
checks.
        lda SPRITE_POS_X_DELTA          ; if not lined up on the
rope we get a false positive
        cmp #4                          ; for collisions around
a 'rope hole'
        beq @end                        ; If we are lined up,
and blocked, its' solid ground

        bcc @deltaLess                  ; if less than 4 - shift
left one
        jsr MovePlayerLeft
        rts
@deltaLess                              ; Otherwise shift right
one
        jsr MovePlayerRight
        rts

@goRopeClimb
        lda #PLAYER_STATE_ROPE
        jmp ChangePlayerState

@noRope
@end
        rts

IDLE_VAR
        byte $00
#endregion
;=================================================================
====================================
;
STATE WALKING
```

```
;-------------------------------------------------------------
-----------------------------------
#region "Player State Walking Right"
PlayerStateWalkR
        lda PLAYER_SUBSTATE
        bne @running
        ;-------------------------------------------------------
SETUP CODE GOES HERE
        ldx #0                                  ; Use sprite
number 0
        lda #<ANIM_PLAYER_WALK_R                ; load animation
in ZEROPAGE_POINTER_1
        sta ZEROPAGE_POINTER_1
        lda #>ANIM_PLAYER_WALK_R
        sta ZEROPAGE_POINTER_1 + 1

        jsr InitSpriteAnim                      ; initialize the
animation
        lda #PLAYER_SUBSTATE_RUNNING            ; set substate
to RUNNING
        sta PLAYER_SUBSTATE
        rts                                     ; wait till next
frame to start
        ;-------------------------------------------------------
----------------------
@running
        jsr JoystickReady
        beq @input                          ; Check creates the
'fix' pause for scroll resetting
        rts
@input
@gravity

        ldx #0                              ; Rope Check
        ;jsr CheckBlockUnder
        lda COLLIDER_ATTR
        cmp #COLL_ROPE
        beq @joyCheck

        jsr ApplyGravity                    ; Apply Gravity - if we
are not falling
        bne @joyCheck                       ; check the joystick
input

        inc PLAYER_FALLCOUNT
        lda PLAYER_FALLCOUNT
        cmp #10
```

```
        bcs @falling
        rts

@falling
        lda #PLAYER_STATE_FALL          ; Otherwise change to
falling state
        jmp ChangePlayerState

@joyCheck
        lda #0
        sta PLAYER_FALLCOUNT

        lda JOY_X
        beq @idle                       ; if JOY_X is 0 we are
idling and need to change states
        bmi @idle                       ; if negative we are
idling

        jsr MovePlayerRight             ; Move player one pixel
across - A = move? 0 or 1
        beq @doneJoy                    ; move code clear, we
can continue

        lda COLLIDER_ATTR
        cmp #COLL_STAIR
        bne @doneJoy
                                        ; Switch to climb stairs
here
        lda #PLAYER_STATE_STAIRS_R
        jmp ChangePlayerState

@doneJoy
        rts
@idle
        lda #PLAYER_STATE_IDLE
        jmp ChangePlayerState

#endregion

#region "Player State Walking Left"
PlayerStateWalkL
        lda PLAYER_SUBSTATE
        bne @running
        ;-------------------------------------------------------
SETUP CODE GOES HERE
        ldx #0                                  ; Use sprite
number 0
```

```
        lda #<ANIM_PLAYER_WALK_L                ; load animation
in ZEROPAGE_POINTER_1
        sta ZEROPAGE_POINTER_1
        lda #>ANIM_PLAYER_WALK_L
        sta ZEROPAGE_POINTER_1 + 1

        jsr InitSpriteAnim                      ; initialize the
animation
        lda #PLAYER_SUBSTATE_RUNNING            ; set substate
to RUNNING
        sta PLAYER_SUBSTATE
        rts                                     ; wait till next
frame to start
        ;-------------------------------------------------------
----------------------
@running
        jsr JoystickReady
        beq @input                          ; check to do the 'fix'
pause for scroll reset
        rts
@input

        ldx #0                              ; Rope Check
        ;jsr CheckBlockUnder
        lda COLLIDER_ATTR
        cmp #COLL_ROPE
        beq @joyCheck

        jsr ApplyGravity                    ; we are falling
        bne @joyCheck

        inc PLAYER_FALLCOUNT                ; if we fall > 8 pixels
we are truely falling
        lda PLAYER_FALLCOUNT
        cmp #10
        bcs @falling

        rts

@falling
        lda #PLAYER_STATE_FALL
        jmp ChangePlayerState

@joyCheck
        lda #0
        sta PLAYER_FALLCOUNT
        lda JOY_X
```

```
        bpl @idle                       ; if it's 0 or greater,
it's back to idle

        jsr MovePlayerLeft              ; move the player one
pixel left
        beq @doneJoy                    ; if move code is clear,
we can continue

        lda COLLIDER_ATTR               ; check if our collider
was a stair
        cmp #COLL_STAIR
        bne @doneJoy

        lda #PLAYER_STATE_STAIRS_L      ; if it is, change state
to climb stairs
        jmp ChangePlayerState


@doneJoy
        rts

@idle
        lda #PLAYER_STATE_IDLE
        jmp ChangePlayerState
#endregion
;=================================================================
==================================
;
STATE FALLING
;-----------------------------------------------------------------
----------------------------------
;  A base falling state
;-----------------------------------------------------------------
----------------------------------
#region "PlayerStateFall"
PlayerStateFall
        lda PLAYER_SUBSTATE                     ; test for first
run
        bne @running
        ;-------------------------------------------------------
SETUP CODE GOES HERE
        ldx #0                                  ; Use sprite
number 0
        lda #<ANIM_PLAYER_FALL                          ; load
animation in ZEROPAGE_POINTER_1
        sta ZEROPAGE_POINTER_1                  ; when falling
we don't want to immediately
```

```
        lda #>ANIM_PLAYER_FALL                          ; go to a
falling animation
        sta ZEROPAGE_POINTER_1 + 1

        jsr InitSpriteAnim                              ; initialize the
animation

        lda #PLAYER_SUBSTATE_RUNNING                    ; set substate
to RUNNING
        sta PLAYER_SUBSTATE
        rts                                             ; wait for the
next frame to take effect
        ;-------------------------------------------------------
----------------------
@running
        ;-------------------------------------------------------
--- JOYSTICK INPUT
        jsr JoystickReady
        beq @input                                      ; not ready for
input
        rts
@input
                                                        ; process valid
joystick input
                                                        ; for falling
there is no input
        jsr ApplyGravity
        ;-------------------------------------------------------
--------------------
        bne @idle
        rts
@idle
        lda #PLAYER_STATE_IDLE
        jmp ChangePlayerState
#endregion
;=============================================================
====================================
;
STATE STAIRS RIGHT
;-------------------------------------------------------------
-----------------------------------
;  Player state for climbing stairs
;-------------------------------------------------------------
-----------------------------------
#region "PlayerStateStairsR"
PlayerStateStairsR
```

```
        lda PLAYER_SUBSTATE                     ; test for first
run
        bne @running
        ;-------------------------------------------------------
SETUP CODE GOES HERE
                                ; TODO - some check to change to
walking right animation
                                ;        if it's currently
different

;        ldx #0                                  ; Use sprite
number 0
;        lda #<ANIM_TEST                         ; load
animation in ZEROPAGE_POINTER_1
;        sta ZEROPAGE_POINTER_1
;        lda #>ANIM_TEST
;        sta ZEROPAGE_POINTER_1 + 1

;        jsr InitSpriteAnim                      ; initialize
the animation
        lda #PLAYER_SUBSTATE_RUNNING            ; set substate
to RUNNING
        sta PLAYER_SUBSTATE
        rts                                     ; state change
goes into effect next frame
        ;-------------------------------------------------------
----------------------
@running
        ;-------------------------------------------------------
--- JOYSTICK INPUT
        jsr JoystickReady
        beq @input                              ; not ready for
input
        rts
@input
        lda JOY_X
        beq @vert_check                         ; X axis in 0 -
check for up
        bmi @idle                               ; if it's -1
(left) return to idle
                                                ; so it has to
be 1 (right) - climb the stair
@vert_check
                                                ; TO DO : check
for an up press

@climb
```

```
                                                ; check for
stair collider
        jsr MovePlayerRight                     ; attempt to
move right
        lda COLLIDER_ATTR                       ; we can't -
because of a stair
        cmp #COLL_STAIR
        beq @go_up                              ; so go up
instead

        lda #PLAYER_STATE_WALK_R                ; else go back
to walking (top of stair)
        jmp ChangePlayerState

@go_up
        jsr MovePlayerUp
        rts
@idle
        lda #PLAYER_STATE_IDLE                   ; return to
idle (which will likely go to fall)
        jmp ChangePlayerState
        ;-------------------------------------------------------
------------------
#endregion

;=================================================================
==================================
;
STATE STAIRS LEFT
;-----------------------------------------------------------------
-----------------------------------
;  Player state for climbing stairs left
;-----------------------------------------------------------------
-----------------------------------
#region "PlayerStateStairsL"
PlayerStateStairsL
        lda PLAYER_SUBSTATE                     ; test for first
run
        bne @running
        ;-------------------------------------------------------
SETUP CODE GOES HERE
;        ldx #0                                  ; Use sprite
number 0
;        lda #<ANIM_TEST                         ; load
animation in ZEROPAGE_POINTER_1
;        sta ZEROPAGE_POINTER_1
;        lda #>ANIM_TEST
```

```
;        sta ZEROPAGE_POINTER_1 + 1

;        jsr InitSpriteAnim                         ; initialize
the animation
        lda #PLAYER_SUBSTATE_RUNNING            ; set substate
to RUNNING
        sta PLAYER_SUBSTATE
        ;-------------------------------------------------------
----------------------
@running
        ;-------------------------------------------------------
--- JOYSTICK INPUT
        jsr JoystickReady
        beq @input                                ; not ready for
input
        rts
        ;-------------------------------------------------------
-----------------
@input                                            ; process valid
joystick input
        lda JOY_X
        beq @vert_check                           ; TODO - check
for vert joystick (up)
        bpl @idle                                 ; if it's 0 or
1, go to idle


@vert_check


@climb                                            ; check for
stair collider
        jsr MovePlayerLeft                       ; attempt to move
right
        lda COLLIDER_ATTR                        ; check for stair
char to the right
        cmp #COLL_STAIR
        beq @go_up                                ; if is, climb
it (go up)

        lda #PLAYER_STATE_WALK_L                  ; it's clear (or
something else) so change state
        jmp ChangePlayerState                     ; to walking
left

@go_up
        jsr MovePlayerUp
```

```
        rts

@idle
        lda #PLAYER_STATE_IDLE                  ; return to idle
state
        jmp ChangePlayerState
        ;-------------------------------------------------------
----------------


        rts
#endregion

;=============================================================
==================================
;
STATE ROPE UP
;-------------------------------------------------------------
----------------------------------
;  Climbing a rope up
;-------------------------------------------------------------
----------------------------------
#region "PlayerStateRopeUp"
PlayerStateRope
        lda PLAYER_SUBSTATE                     ; test for first
run
        bne @running
        ;-------------------------------------------------------
SETUP CODE GOES HERE
        ldx #0                                  ; Use sprite
number 0
        lda #<ANIM_CLIMB_ROPE_UP                ; load animation
in ZEROPAGE_POINTER_1
        sta ZEROPAGE_POINTER_1
        lda #>ANIM_CLIMB_ROPE_UP
        sta ZEROPAGE_POINTER_1 + 1

        jsr InitSpriteAnim                      ; initialize the
animation
        lda #PLAYER_SUBSTATE_RUNNING            ; set substate
to RUNNING
        sta PLAYER_SUBSTATE
        rts                                     ; change takes
effect next frame
        ;-------------------------------------------------------
----------------------
@running
```

```
        ;-------------------------------------------------------
--- JOYSTICK INPUT
        jsr JoystickReady
        beq @input                              ; not ready for
input
        rts
        ;-------------------------------------------------------
-------------------
                                                ; Process valid
joystick input
@input
        lda JOY_X
        beq @vertCheck
        bmi @left
        bpl @right
        rts

@right
        ldx #0
        jsr CheckMoveRight
        beq @goRight
        rts

@goRight
        lda #PLAYER_STATE_WALK_R
        jmp ChangePlayerState

@left
        ldx #0
        jsr CheckMoveLeft
        beq @goLeft
        rts

@goLeft
        lda #PLAYER_STATE_WALK_L
        jmp ChangePlayerState

@vertCheck
                                                ; lets auto
align the player to the rope
                                                ; so they can
pass through 'holes'
        ldx #0
        lda SPRITE_POS_X_DELTA,x
        cmp #4                                  ; they pass
through if delta is 4
        beq @check
```

```
        bcc @less                                   ; if less than
4, shift right one pixel

        jsr MovePlayerLeft                          ; not equal, not
less, must be more - shift left one
        jmp @check
@less
        jsr MovePlayerRight

@check
        lda JOY_Y
        beq @end
        bmi @up
        bpl @down

        rts
@up
        lda #1
        sta SPRITE_ANIM_PLAY            ; play our animation

        jsr MovePlayerUp
        rts
@down
        lda #1
        sta SPRITE_ANIM_PLAY
        jsr MovePlayerDown
        bne @endclimb
        rts

@endclimb
        lda SPRITE_POS_X_DELTA
        cmp #4
        beq @stopClimb
        rts

@stopClimb
        lda #PLAYER_STATE_IDLE
        jmp ChangePlayerState

@end
        lda #0
        sta SPRITE_ANIM_PLAY            ; pause our animation
        ;-------------------------------------------------------
-------------------
        rts
#endregion
```

```
;=================================================================
==================================
;
STATE FRAMEWORK
;-----------------------------------------------------------------
----------------------------------
;  A blank state template to make adding new states easier
;-----------------------------------------------------------------
----------------------------------
#region "Player State Framework"
PlayerState_Framework
        lda PLAYER_SUBSTATE                     ; test for first
run
        bne @running
        ;-------------------------------------------------------
SETUP CODE GOES HERE
        ldx #0                                  ; Use sprite
number 0
        lda #<ANIM_TEST                         ; load animation
in ZEROPAGE_POINTER_1
        sta ZEROPAGE_POINTER_1
        lda #>ANIM_TEST
        sta ZEROPAGE_POINTER_1 + 1

        jsr InitSpriteAnim                      ; initialize the
animation
        lda #PLAYER_SUBSTATE_RUNNING            ; set substate
to RUNNING
        sta PLAYER_SUBSTATE
        rts                                     ; change takes
effect next frame
        ;-------------------------------------------------------
----------------------
@running
        ;-------------------------------------------------------
--- JOYSTICK INPUT
        jsr JoystickReady
        beq @input                              ; not ready for
input
        rts
        ;-------------------------------------------------------
------------------
                                                ; Process valid
joystick input
@input
;        lda JOY_X
;        beq @vertCheck
```

```
;         bmi @left
;         bpl @right
;         rts
@right
@left

@vertCheck
;         lda JOY_Y
;         beq @end
;         bmi @up
;         bpl @down
;         rts
@up
@down

@end
         ;-------------------------------------------------------
-------------------
         rts
#endregion

;=================================================================
==================================
;=================================================================
==================================
;
PLAYER DATA
;-----------------------------------------------------------------
----------------------------------

PLAYER_DATA

PLAYER_STATE                              ; Current state -
walking, standing, dying, climbing
         byte 0
PLAYER_SUBSTATE
         byte 0                           ; Current substate -
jumping up, falling down, start dying
PLAYER_DIRECTION_X
         byte 0
PLAYER_DIRECTION_Y
         byte 0
PLAYER_FALLCOUNT
         byte 0                           ; Counting frames to
actual fall
;-----------------------------------------------------------------
----------------------------------
```

```
;
TABLES/DATA
;------------------------------------------------------------------
------------------------------------
```