```
;=================================================================
==================================
;
COLLISION ROUTINES
;=================================================================
==================================
;
Peter 'Sig' Hewett
;
- 2016,2017
;-----------------------------------------------------------------
---------------------------------
; Routines dealing with collisions between game objects
;
;=================================================================
==================================
;
SPRITE TO BACKGROUND CHARACTERS
;=================================================================
==================================
; Checks to see if the sprite is colliding with a background
character.
; Many of these checks will be 'forward looking' (especially in
movement checks)
; We will be looking to where the sprite will be, not where it
is, and then
; letting the sprite handling routines update the positions and
deltas for us
; if we move.
;
; The original 'framework code' worked against a single screen
and returned a simple
; 1 for blocked and 0 for clear. We need a much better system
for the game engine
; that will handle double buffered screens as well as
calculating screen scrolling.
; It will also need to return more information dealing with
different types of blocks
;
;
;=================================================================
==================================
;
CHECK MOVE LEFT
;-----------------------------------------------------------------
---------------------------------
```

```
; Checks ahead to see if this sprite can love left, of it's
going to be stopped by a blocking
; character
;
; X = sprite we want to check for
;
; Returns A = 0 we can move or A = 1 we are blocked
;
; Modifies A, Y, PARAM1, PARAM2
; X register is left intact
;-------------------------------------------------------------
-----------------------------------
#region "CheckMoveLeft"
CheckMoveLeft
        lda SPRITE_CHAR_POS_X,x                 ; PARAM1 will
hold the adjusted X position
        sta PARAM1
        lda SPRITE_CHAR_POS_Y,x                 ; PARAM2 will
hold the adjusted Y position
        sta PARAM2

        lda SPRITE_POS_X_DELTA,x                ; Load X pos
delta
        sec
        sbc SCROLL_COUNT_X                      ; adjust the
delta for scrolling left/right
        bpl @noAdjX                             ; check to see
if delta wraps to -1 ($FF)

        dec PARAM1                              ; if the delta
wraps, adjust the X char position


@noAdjX
        and #%0111                              ; mask the delta
value back to a 0-7 value
        cmp #0                                  ; Check X delta
to see if the sprite is 'flush'
        beq @checkLeft                          ; if it is,
continue
        lda #0                                  ; else return
with a 'clear' code
        rts

@checkLeft
        lda SPRITE_POS_Y_DELTA,x                ; load the
sprite delta Y pos
```

```
        sec                                     ; subtract the
current Y scroll
        sbc SCROLL_COUNT_Y
        bpl @noAdjY                             ; if it wraps,
decrement the Y char position

        dec PARAM2
@noAdjY
        and #%0111                              ; mask back to a
0-7 value
        beq @checkLeft2                         ; if adjusted
delta Y is 0, we only need to check 2
                                                ; characters

                                                ; If not we need
to check 3 characters

        ldy PARAM2                              ; load adjusted
Y char position (screen line)
        iny                                     ; increment by
one line

        jsr FetchPlayfieldLineAddress           ; Use
screen_routines helper to fetch the address

        ldy PARAM1                              ; fetch sprites
X position
        dey                                     ; sub 1
character (left)

;        lda PARAM1                              ; fetch sprites
X character position
;        clc
;        adc #39                                 ; add 39, go
down one line and left one character
;        tay

        lda (ZEROPAGE_POINTER_1),y              ; fetch the
character from screen memory

        jsr TestBlocking                        ; test for a
blocking character
        bne @blockedLeft

                                                ; otherwise
check the other 2 characters
@checkLeft2
```

```
        ldy PARAM2                                      ; fetch the
sprite Y char position
;        ldy SPRITE_CHAR_POS_Y,x
        dey                                             ; go one line up
        jsr FetchPlayfieldLineAddress

        ldy PARAM1
        dey                                             ; one char left


        lda (ZEROPAGE_POINTER_1),y
        jsr TestBlocking
        bne @blockedLeft

        tya
        clc
        adc #40
        tay

        lda (ZEROPAGE_POINTER_1),y
        jsr TestBlocking
        bne @blockedLeft

        lda #0
        rts

@blockedLeft
        lda #1
        rts

#endregion
;-------------------------------------------------------------------
----------------------------------

;===================================================================
==================================
;
CHECK MOVE RIGHT
;-------------------------------------------------------------------
----------------------------------
; Based on the previous 'CanMoveRight' routine. It checks ahead
to see if there are going to be
; blocking characters in the direction we want to move.
;
; This version checks the appropriate front screen and returns a
code. 0 for clear or 1-255 for
; any special action that can or must be taken.
```

```
;
; X = Sprite to check against
;
; Modifies Y, PARAM1, PARAM2
;
; Returns A = blocking code
; X register is left intact
;-------------------------------------------------------------------
-----------------------------------

#region "CheckMoveRight"
CheckMoveRight

        lda SPRITE_CHAR_POS_X,x         ; PARAM1 will hold the
adjusted X position
        sta PARAM1
        lda SPRITE_CHAR_POS_Y,x         ; PARAM2 will hold the
adjusted Y position
        sta PARAM2

        lda SPRITE_POS_X_DELTA,x        ; Fetch X Delta
        sec
        sbc SCROLL_COUNT_X              ; Subtract the scroll
count
        bpl @noAdjX

        dec PARAM1                      ; increment PARAM1

@noAdjX
        and #%0111                      ; mask back to a range
of 0-7                                  ; No adjustment is
needed
        cmp #0                          ; is new delta = 0?
        beq @checkRight                 ; if so, we can check
right
        lda #0                          ; otherwise give a clear
code and carry on
        rts


@checkRight
        lda SPRITE_POS_Y_DELTA,x        ; Load the Y delta
        sec
        sbc SCROLL_COUNT_Y              ; adjust by the current
Y scroll value
        bpl @noAdjY                     ; if the delta goes
below 0 adjust Y position
```

```
        dec PARAM2

@noAdjY
        and #%0111                      ; mask adjusted delta
value back to 0 - 7
                                        ; if Y delta is 0, we
are flush on the Y axis, so only
        beq @rightCheck2                ; check 2 characters
                                        ; Otherwise we are
overlapping, so we need to check 3
        ldy PARAM2                      ; Fetch the sprites Y
character position
        iny                             ; add 1 (one character
line down)

        jsr FetchPlayfieldLineAddress   ; fetch the address for
the start of that screen line

        ldy PARAM1                      ; fetch the sprites X
position and store it in Y
        iny                             ; add 1 character
(right)
        lda (ZEROPAGE_POINTER_1),y      ; fetch the screen
character
        jsr TestBlocking                ; see if it's a blocking
character
        bne @blockedRight               ; return if it is

@rightCheck2                            ; check the 2 characters
to the right of the sprite
;        ldy SPRITE_CHAR_POS_Y,x         ; get the sprite Y
character coord
        ldy PARAM2
        dey                             ; subtract one (one
character down)

        jsr FetchPlayfieldLineAddress   ; fetch the address for
the start of that line

        ldy PARAM1                      ; Load the adjusted X
character position in Y
        iny                             ; add one character to
the right

        lda (ZEROPAGE_POINTER_1),y      ; fetch the character
```

```
        jsr TestBlocking                ; test the character for
blocking
        bne @blockedRight               ; exit and return the
code if blocking

        tya                             ; Add #40 to the current
X position
        clc                             ; this is the same as
going down one block
        adc #40
        tay
        lda (ZEROPAGE_POINTER_1),y      ; load the character to
CheckMoveDown
        jsr TestBlocking                ; test it
        bne @blockedRight               ; return the code if
blocking

        lda #0                          ; else return with a
'clear' code
        rts

@blockedRight
        lda #1
        rts


#endregion
;-------------------------------------------------------------------
-----------------------------------
;===================================================================
===================================
;
CHECK MOVE DOWN
;-------------------------------------------------------------------
-----------------------------------
;
; X = sprite we want to check for
;
; Returns : A = 0 we move or A = 1 we are blocked
;
; Modifies : Y,PARAM1, PARAM2
;            X is left intact
;-------------------------------------------------------------------
-----------------------------------
#region "CheckMoveDown"
CheckMoveDown
```

```
        lda SPRITE_CHAR_POS_X,x                 ; Fetch the
sprites X character coord
        sta PARAM1                              ; PARAM1 will
hold the adjusted Y coord
        lda SPRITE_CHAR_POS_Y,x                 ; Fetch the
sprites Y character coord
        sta PARAM2                              ; PARAM2 will
hold the adjusted Y coord


                                                ; Adjust the Y
Delta and Pos Y values
        lda SPRITE_POS_Y_DELTA,x                ; load the delta
Y
        sec
        sbc SCROLL_COUNT_Y                      ; subtract the Y
scroll count
        bpl @noAdjustY                          ; if it's less
than 0, adjust Y pos

        dec PARAM2


@noAdjustY
        and #%0111                              ; mask back to
0-7 value
                                                ; if adjusted
delta Y is 0, we are flush
        beq @downCheck                          ; and do a
character check
        lda #0                                  ; else return a
clear code
        rts


@downCheck
        lda SPRITE_POS_X_DELTA,x                ; check the X
delta, if it's flush (0) we only
        sec
        sbc SCROLL_COUNT_X
        bpl @noAdjX

        dec PARAM1

@noAdjX
        and #%0111
```

```
        beq @downCheck2                         ; need to check
2 characters

        ldy PARAM2                              ; load the
adjusted Y character line
        iny                                     ; increment to
one line down

        jsr FetchPlayfieldLineAddress
                                                ; load sprite X
char position
        ldy PARAM1
        iny                                     ; inc X pos
(left character)
        lda (ZEROPAGE_POINTER_1),y              ; fetch the
character

        jsr TestBlocking
        bne @downBlocked

@downCheck2
        ldy PARAM2                              ; load sprite Y
char coord
        iny                                     ; increment down
one line

        jsr FetchPlayfieldLineAddress

        ldy PARAM1
        lda (ZEROPAGE_POINTER_1),y

        jsr TestBlocking
        bne @downBlocked

        lda #0
        rts

@downBlocked
        lda #1
        rts
#endregion

;=================================================================
====================================
;
CHECK MOVE UP
```

```
;-------------------------------------------------------------
-----------------------------------
; Checks ahead to see if this sprite can move up, or if it's
going to be stopped by a blocking
; character
;
; X = sprite we want to check for
;
; returns A = 0 we can move or A = 1 we are blocked
;
; Modifies Y,PARAM1,PARAM2
;-------------------------------------------------------------
-----------------------------------
#region "CheckMoveUp"
CheckMoveUp
        lda SPRITE_CHAR_POS_X,x                 ; load sprites X
character pos
        sta PARAM1                              ; adjusted X
char pos will be in PARAM1
        lda SPRITE_CHAR_POS_Y,x                 ; adjusted Y
character pos
        sta PARAM2                              ; load sprites y
character pos

        lda SPRITE_POS_Y_DELTA,x                ; load sprites Y
delta
        sec                                     ; subtract Y
scroll value
        sbc SCROLL_COUNT_Y
        bpl @noAdjY                             ; if it wraps
past 0 - adjust Y char pos

        dec PARAM2

@noAdjY
        and #%0111                              ; mask back to a
value of 0-7
        beq @checkUp                            ; if it's 0 - we
are flush and do a check
        lda #0                                  ; otherwise
return with a 'clear' code
        rts

@checkUp
        lda SPRITE_POS_X_DELTA,x                ; load the
sprite X delta value
        sec
```

```
        sbc SCROLL_COUNT_X                      ; subtract
current scroll X value
        bpl @noAdjX                             ; if it wraps
past 0 - adjust the X char pos

        dec PARAM1

@noAdjX
        and #%0111                              ; mask back to a
0-7 value
        beq @checkUp2                           ; if we are not
flush we need to check 2 characters

        ldy PARAM2                              ; fetch the
adjusted sprite Y char coord
        dey                                     ; subtract 2
lines (up)
        dey

        jsr FetchPlayfieldLineAddress

        ldy PARAM1                              ; load adjusted
X character pos
        iny                                     ; inc X by 1
(one char right)
        lda (ZEROPAGE_POINTER_1),y


        ;jsr TestBlocking
        bne @upBlocked


@checkUp2
        ldy PARAM2                              ; load the
adjusted sprite char Y position
        dey                                     ; decrement (go
up) by 2 lines
        dey

        jsr FetchPlayfieldLineAddress

        ldy PARAM1                              ; load adjusted
X character position

        lda (ZEROPAGE_POINTER_1),y
        ;jsr TestBlocking
        bne @upBlocked
```

```
        lda #0
        rts

@upBlocked
        lda #1
        rts

#endregion

;=====================================================================
;
CHECK BLOCK UNDER
;---------------------------------------------------------------------
; Check the block under the sprite for collision (usefull for
things like ropes and chests)
;---------------------------------------------------------------------
#region "CheckBlockUnder"
CheckBlockUnder
        lda SPRITE_CHAR_POS_X,x         ; PARAM1 will hold the
adjusted X position
        sta PARAM1
        lda SPRITE_CHAR_POS_Y,x         ; PARAM2 will hold the
adjusted Y position
        sta PARAM2

        lda SPRITE_POS_X_DELTA          ; Fetch X delta
        sec
        sbc SCROLL_COUNT_X              ; subtract the scroll
count
        clc
        adc #3
        bpl @noAdjX                     ; if it's 0 or greater,
don't adjust X position
        dec PARAM1                      ; otherwise subtract 1

@noAdjX
        and #%0111                      ; Mask delta back to a
value between 0-7
        beq @adjY                       ; we are flush on the X
axis - do check on Y
                                        ; Otherwise ?? I could
do another set of lookups
                                        ; based on 'non flush'
```

```
@adjY
        lda SPRITE_POS_Y_DELTA          ; Fetch the Y delta
        sec
        sbc SCROLL_COUNT_Y
        bpl @noAdjY
        dec PARAM2

@noAdjY
        and #%0111                      ; if we are flush on Y
we only need to check 2 characters
        beq @underCheck2                ; otherwise we need to
check 3

@underCheck
        ldy PARAM2                      ; fetch the adjusted Y
position
        iny                             ; add one line (down)

        jsr FetchPlayfieldLineAddress   ; Fetch the address for
the start of the screen line
                                        ; returned in
ZEROPAGE_POINTER_1
        ldy PARAM1                      ; fetch the sprites
adjusted X character position
        lda (ZEROPAGE_POINTER_1),y      ; fetch the screen
character
        jsr TestBlocking
        bne @blockingUnder              ; exit if its a blocking
char (shouldn't be)
                                        ; TO-DO exit if it's a
special character to keep
                                        ; our 'last test' intact
        lda COLLIDER_ATTR
        bne @special_under               ; A special block is
under us - exit

@undercheck2
        ldy PARAM2                      ; fetch the adjusted Y
position
        dey                             ; one line up
        jsr FetchPlayfieldLineAddress   ; Fetch the address for
the start of the screen line
                                        ; returned in
ZEROPAGE_POINTER_1
```

```
        ldy PARAM1                      ; fetch the sprites
adjusted X character position
        lda (ZEROPAGE_POINTER_1),y
        jsr TestBlocking
        bne @blockingUnder
        lda COLLIDER_ATTR
        bne @special_under

        tya
        clc
        adc #40
        tay
        lda (ZEROPAGE_POINTER_1),y
        bne @blockingUnder
        lda COLLIDER_ATTR
        bne @special_under
        lda #0
        rts
@blockingUnder
        lda #1
        rts
@special_under          ; return the 'special - not 1 or 2'
        rts




        rts
#endregion
;====================================================================
====================================
;
TEST CHARACTER FOR BLOCKING
;====================================================================
====================================
; Originally we had a simple check for blocking characters >
128. Using Charpad
; we can test for attributes we encode in the upper half byte of
the color info
; Note : it seems you can't read this back directly from color
ram, but we can
;        look it up easily enough given the character number.
;
;
;
; A = Character number we're checking against
```

```
; Returns: A = 0 or 1 (clear - blocked) and stores the collison
attribute so we can test against it
;
; Modifies A
; Restores X, Y
;-------------------------------------------------------------
-----------------------------------
;
COLLIDER CODES
;-------------------------------------------------------------
-----------------------------------
COLL_CLEAR = $00
COLL_BLOCK = $10
COLL_STAIR = $20
COLL_ROPE  = $30
;-------------------------------------------------------------
-----------------------------------
#region "TestBlocking"
TestBlocking
        sta COLLIDER_ATTR       ; save the info passed to us, we
need to use A

        txa                     ; store X and Y on the stack
        pha                     ; the routines we go back to
need these intact
        tya
        pha

        ldx COLLIDER_ATTR       ; load the character number in X
        lda ATTRIBUTE_MEM,x     ; fetch the attribute
        and #%11110000          ; mask the color info - leaving
the attribute
        sta COLLIDER_ATTR       ; store it so the rest of the
program can use it
        beq @returnClear        ; 0 is always clear - so return
        cmp #COLL_ROPE
        beq @returnClear        ; Ropes ($30) don't block

        bne @blocking           ; Blocking character


@returnClear
        pla                     ; restore x and y off the stack
        tay
        pla
        tax
```

```
        lda #0                  ; return 0 - A clear code
        rts

@blocking
                                ; ATM only basic collide info -
if not 0, we're blocking
        pla                     ; restore X and Y from the stack
        tay
        pla
        tax

        lda #1                  ; set to blocking and return

        rts

COLLIDER_ATTR                  ; A place to store our attribute,
and read it later if needed
        byte 0

#endRegion
```