DOCUMENTATION OF VARIABLES USED IN LEAPFROG

NOTE: Some memory locations are used for more than one purpose, particularly for use in the introductory title and then in the main game. They will be named differently in each section.

DURING INTRODUCTORY TITLE

1F57 **HC_VOLTMACE** ;horizontal position of "VO" in intro
@002f initialised to $C0
@0086 decremented by 4
@017d output to object1HCB "VO" ; others are set arithmetically with fixed offsets

1F58 **HC_by** ; horizontal position of "by" in intro
@002f initialised to $D8
@008e incremented by 4
@00e2 output to object1HCB "by"

1F59 **brief_pause_at_startup** ; counts 64 cycles before animation begins
initialised to $40 @002f
tested for 0 @003d
decremented @006b

1F5A **bounce_index**
initialised to $19 @002f
tested for 0 @005a
decremented @0072
index to horiz_pos_LEAPFROG_1 @00a3
index to inc_horiz_pos_LEAPFROG_23@00a9
index to inc_horiz_pos_LEAPFROG_23@00af
index to inc_horiz_pos_LEAPFROG_4 @00b5
index to vert_pos_LEAPFROG_1234 @00bb
bits %00000111 used for bouncy sound
compared to $11 @00f5
@0137 if=0, causes LEAPFROG to display at largest size, %10
@013a if<9, causes LEAPFROG to display at size %01
otherwise LEAPFROG displays at size %00

1F5B **byVOLTMACE_timer** ; decrementing counter during “by VOLTMACE” animation
@002F initialised to $20
@005F tested for 0
@007E decremented

1F5C NOT USED
initialised to $00 @002f

1F5D **pause_at_end**
@002F initialised to $80

@0064 tested for 0
@0095 decremented

1F6A **animation_delay**
@0027 initialized to $05
@0044 decremented, and tested for 0
@004C set to $02 (during “by VOLTMACE” animation)
@0077 set to $04 (during bouncing animation)

DURING GAME PLAY

1F0E **copy_interobject**
@02C5 saved from PVI
@036C tested for frog collision
@0523 tested for frog collision
@0BFE tested isubr_test_interobject

1F0F not_found, and no activity seen using the WinArcadia PVI monitor. This may possibly have been something I used

1F1E **stopwatch**

This is a timer that gets 'decremented' approx every five seconds. The 'decrement' is actually a shift left, so it counts down FF,FE,FC,F8,F0,E0,C0,80,00 at which point life is lost. When a run is completed by the frog landing safely in a hole, the number of 1's are counted and the score incremented for each of them. It is initialised at the start of each run, but allows a shorter time for each run as the holes are filled.

(Its an odd way of implementing this, probably driven by its direct use as a bar at the bottom of the screen to indicate time left)

@02D5 (start_new_frame) if == 0, goto frog_mishap_2
@03BB (start_fresh_run) loaded from LUT_time_allowed[x].
= F8,FC,FE,FF depending on the number of holes left unfilled.
@05A3 (safely_home) shifted left and bit 7 tested
DOES SOMETHING TO SCORE
Doesn't seem to be saved
@09CF (near end of screen output) if timer_P == 0 (~every 5s) then stopwatch<<
@0A3D output as a bar in the time left display
@0E48 (subr_init_PVI) initialised to $FF at same time as lifes_left = $33

1F1F **lifes_left**

7	6	5	4	3	2	1	0
	Player 1				Player 2		

@03FB $10 subtracted from it, then.....
@03fe bits 654 are tested for zero

@0413 bits 210 are tested
@0423 decremented
@0426 bits 210 are tested
@043e bits 654 are tested for zero
@05CA clear bits 76543
@05FA (bonus_life_and_score) test bit 6 (check for max lives)
@05FE (bonus_life_and_score) add $10 to it (give player 1 an extra life)
@0602 (bonus_life_and_score) test bit 2 (check for max lives)
@0606 (bonus_life_and_score) add $01 to it (give player 2 an extra life)
@0A1D (display_life_counter) read into r1
@0A2B if bit3 of control_bits_2 == 0, r1 is shifted right four times
@0A32 ...and bits 76543 are cleared
@0A34 ...and is used as index to shape of life display.
@0E4D (subr_init_PVI) initialised to $33 at same time as stopwatch= $ff

(During attract mode, it changes 33, 23, 13, 03 when a life is lost)

START OF 32 BYTES SCRATCH

1F4E **timer_P**

timer_P is decremented every frame during both gameplay and select player.
@027d initialised to $00
@03C1 (start_fresh_run) timer_P = timer_P_reset_value
@058E (score_for_speed) compared to $32 (50)
@0594 (score_for_speed) compared to $64 (100)
@059A (score_for_speed) compared to $96 (150)
@09CA (unconditional, during display scan) **decremented**, and if == 0 then.....
@09D7 (unconditional, during display scan) .....new value saved from timer_P_reset_value

In this mode, timer_P counts one second, and then decrements timer_Q
@0C2D (subr_select_start) set to $32 (one second timer)
@0C54 (subr_select_start) **decremented** ; if == 0, reset to $32 (one second timer)
@0D0D (subr_1or2_player) test bit 3 to determine if flashing down arrow in select screen is diplayed

1F4F **timer_Q**

timer_Q is only decremented every frame either after frog reaches a hole or it looses a life.
@027D initialised to $00
@02CF (start_new_frame) tested for 0
@0379 **decremented** , if != 0, goto start_screen_output , else test control_bits and possibly reset.
@0454 set to $FF
@0462 (prepare_for_new_run2) set to $40
@0499 (frog_mishap_2) set to $20
@0569 (safely_home) set to $00 - $07
@05E5 set to $50

timer_Q is only decremented once per second

@0C3B (subr_select_game) set to 5 in attract mode, else $19(25)

@0C68 (subr_select_game) if timer_P == 0, **decrement** timer_Q and if ==0 causes a RESET

@0C73 (subr_select_game) bit 1 used as index to look up value to save in HC_down_arrow

@0C79 (subr_select_game) bit 1 used as index to either score1 or score2

1F50 **array_HC_frogs_in_holes** [0-2]

(~HC for the fours holes: $20, $47, $6F, $98 – there is some lee way)

(The first frog to reach safety goes in [2], the second in [1], the third in [0].
The fourth frog is object 1 and doesn't need storage in the variable.)

@027d, [0-2] initialised to 0

@046F, [0-2] initialised to 0

@0569, [r1-1] set to object1HC

@06DE, output [0-2] to HC objects 2,3,4

HC_down_arrow

@0C76, (subr_select_game) [0] is set to either $20 or $80

@0D05, (subr_1or2_player) [0] sometimes gets transfered to object1HCB

1F53 **array_ HC_A_logs** [0-2]

@027d initialised to [$14 (20), $64 (100), $BC (188)]

@04AB looked up with index

@0DC0 [0] moved to object2HCB, [2] moved to object4HCB

@0DC9 [1] moved to object3HCB

1F56 **array_HC_A_lillies**[0-2]

@027d initialised to [$03, $53(83), $A3(163)]

@04C3 loaded, indexed. So following bytes probably have a second purpose

@0773 loaded

@078A saved

1F59 **array_ HC_B_logs** [0-2]

@027D initialised to [$11 (17), $64(100), $B5 (181)]

@04CE loaded

1F5C **array_HC_B_lillies** [0-2]

@027d initialised to [$2A (42), $52 (94), $AA (170)]

@04ED tested extensively , probably comparing lilyHC to frog pos

@07A5 new value saved

1F5F **array_HC_snake** [0-2]

@027D initialised to $E1 (225), $E4 (228), $F4 (244)

@0826 all three bytes are decremented (moved left).....

@082A ... and if ==0, reset to $E5

ALTERNATE USE, IN THE 'YOU BEAT IT' SEQUENCE :

1F60 **four_cycles** (of sound effect and flashing score 9999)

@0AC9, set to $04
@0B07, decremented, tested for zero

1F61 **frame_counter**
@0AEF, set= effects_counter
@0AFB, decremented and tested in a loop counting frame cycles

1F62 **array_HC_A_trucks** [0-2]
initialised @027d to [$20 (32), $78 (120), $D2 (210)]
@07F8 compared to **max_HC_road**

ALTERNATE USE, IN THE 'YOU BEAT IT' SEQUENCE :
effects_counter
@0ACE 1F62 set to $09
@0AEF this value used to set pitch and frame_counter

1F65 **array_HC_A_cars** [0-2]
@027d initialised to [$02, $64 (100), $B0 (176)
@07E6 decremented and if 0 gets new value from max_HC_road

1F68 **array_HC_B_cars** [0-2]
@027d initialised to [$29 (41), $75 (117), $D7 (215)]
@080E incremented
@0810 compared >= max_HC_road

1F6B **array_HC_B_trucks** [0-2]
@027D initialised to [$44(68), $99(153), $F3(243)]
@0758 read; generally gets decremented to move truck left, but when it reaches 0 a new value is fetched from max_HC_road
@0770 saved

END OF 32 BYTES SCRATCH

START OF 40 BYTES OF VERTICAL BAR DEFINITION

1F80 **grid_v1**

Start of the vertical bar definitions for the grid. Only the first six byte are used for this purpose in game play, the rest are variables.
@028A initialise $28 bytes from initialise_1
1F80
@028A initialised to $00
@0B17 reset to $D0

1F81
@028A initialised to $00
@0B17 reset to $08

1F82

@028A initialised to $84 ; THIS AND THE NEXT BYTE DEFINE THE HOLES
@0B17 reset to $55

1F83

@028A initialised to $21
@0B17 reset to $38

1F84

@028A initialised to $00
@0B17 reset to $10

1F85

@028A initialised to $00
@0B17 reset to $08

1F86 **array_shape_A_logs[0-2]**

@028A initialised to $FF, $FF, $FF
@04A7 read and saved in r2
@0717 compared to log_shape , possibly rotated right and bit 7 set, and saved
@0DA4 (subr_two_bytes_obj234) output to objects 2,3,4

1F89 **status_A_lillies**

@028A initialised to $03
@035D if status_A_lillies == LUT_index_blue_lily[N] goto frog_mishap_2
@04B6 (frog_in_A_lillies) if status_A_lillies == 3, gosub subr_flash_lily_decision
@04BD status_A_lillies= $3, $15, $18 or $1B
@077D compared with LUT_index_blue_lily[r2]
@0A82 compared to 3, (see **index_A_lillies_colours** below)

1F8A **counter_flash_A_lillies**

@028A initialised to $00
@04C0 set to either 0 or $64 (100)
@0A87 if counter_flash_A_lillies == 0, then index_A_lillies_colours = status_A_lillies
@0A8B counter_flash_A_lillies --

1F8B **index_A_lillies_colours**

@028A initialised to $00
@0868 passed to subr_obj_size_colour as the index (via to subr_lillies)
@0A94 case:

if status_A_lillies== 3,	exit with index_A_lillies_colours= 3
if counter_flash_A_lillies == 0,	exit with index_A_lillies_colours= var_fiddle?
if counter_flash_A_lillies {bit3} == 0,	exit with index_A_lillies_colours= var_fiddle?
else	exit with index_A_lillies_colours = 3

1F8C **array_shape_B_logs[0-2]**

@028A initialised to [$FF,$FF,$FF]
@04CA (frog_in_B_logs) array_shape_B_logs[N] passed to end_of_log in r2
@07CA (when exiting left of screen) shifted left on bit to shorten the log
@07DD (when reappearing on right) array_shape_B_logs[N] = log_shape
@0DA4 (subr_two_bytes_obj234) output to objects 2,3,4

1F8F **status_B_lillies**
@028A initialised @028A to $03
@035A compared to $15, $18 or $1B
@04D6 (frog_in_B_lillies), compared to 3
@04E0 (frog_in_B_lillies),
@079A compared to convert_012_to_index[r2]
@079F set to 3
@0A66 compared to 3, if equal, saved to index_B_lillies_colours

1F90 **counter_flash_B_lillies**
@028A initialised to $00
@0351 if counter_flash_B_lillies != 0 then test_frog_position else subr_test_interobject
@04E3 set to $64 as two second timer
@0A6A if equal 0, then index_B_lillies_colours= 3
@0A71 decremented
@0A76 if bit 3 == 0 then index_B_lillies_colours= counter_flash_B_lillies

1F91 **index_B_lillies_colours**
@028A initialised to $00
@08B4 read and passed to subr_lillies and hence to subr_obj_size_colour as the index
@0A7A case:

if status_B_lillies == 3,	exit with index_B_lillies_colours= 3
if counter_flash_B_lillies == 0,	exit with index_B_lillies_colours= status_B_lillies
if counter_flash_B_lillies {bit3} == 0,	exit with index_B_lillies_colours= status_B_lillies
else	exit with index_B_lillies_colours = 3

1F92 **J_group_control** speed of A_logs, B_trucks, A_lillies, B lillies
@028A initialised to $80 (slowest speed, moves 7 times every 16 frames)
@0705 if (J_group_control & byte from LUT_speed_control) == 0 then don't move this frame
@0B9E (sub_level_control) rotated (depending on bits 7 and 1 of byte read from LUT_level_parameters)

1F93 **K_group_control** speed of B_logs, A_cars, A_trucks
@028A initialised to $10 (intermediate speed, moves 10 times every 16 frames)
@07AB if (K_group_control & byte from LUT_speed_control) == 0 then don't move this frame
@0BAD (sub_level_control) rotated (depending on bits 6 and 0 of byte read from LUT_level_parameters)

1F94 **B_car_control**

@028A initialised to $08
@0806 if (B_car_control & byte from LUT_speed_control) != 0, then move B_cars.
@0BC0 (sub_level_control) if bit 5 of LUT_level_parameters[x]==1, B_car_control is rotated right

1F95 **snake_control**

@028A initialised to $01
@081E if snake_control & r3 == 0 then move snake
@0BCA (sub_level_control) if bit 4 of LUT_level_parameters[x] == 1,
snake_control is shifted left, twice

1F96 **array_pointer_B_trucks**[0-2]

used as offset s within the shape definitions of the B_trucks (V O L T M A C E)
@028A initialised to [$40, $48, $00]
@075F value read from array
@076A value tested, manipulated and saved in array
@0D2E (subr_output_objects234_voltmace_trucks)
added to another value and used as an index to a shape

1F99 **frog_on_lily_log**

	7	6	5	4	3	2	1	0
0								
1					B Log <	A Lilly <	B Lilly >	A Log >

@028A initialised to $00
@03AA (start a fresh run) set to $00
@0494 loaded from r1 (offscreen = 0; 1 or 8 if missed a log; 2 or 4 if missed a lily)
@0506 (landed on log or lily) loaded from r1 ; 1 or 8 if log, 2 or 4 if lily
@0537 set to $00
@0670 set to $00
@0732 bits 210 tested
@0739 if bit 2 == 1 move frog left, else bit 0 or 1 must == 1 so move frog right
@07B2 if bit 3 == 1 move frog left

1F9A **joystick_udlr**

	7	6	5	4	3	2	1	0
0								
1					Joystick up	Joystick down	Joystick left	Joystick right

@028A initialised to $00

@02DB bit 3 tested moves frog up
@02E0 bit 2 tested moves frog down
@02E5 bit 1 tested moves frog left
@02EC bit 0 tested moves frog right

@030B (joystick right) set to $01
@0318 (joystick down) set to $04
@032A (joystick left) set to $02
@0337 (joystick up) set to $08

@03AA (start a fresh run) set to $00
@0534 (frog has reached the bank) set to $00
@0673 set to $00
@0A04 (frog_leaping_sound) manipulated and output as pitch

1F9B **frog_animation**

Used as a counter to control animation, movement and soundFX of the frog.

@028A initialised to $00
@032F set to $11
@0310 set to $11 (right)
@031D set to $0F (down)
@032F set to $11 (left)
@033C set to $0F (up)
@03AD (start a fresh run) set to $00
@0618 if == $0D, causes long frog to be loaded
@061C if == 8 causes short frog to be loaded
@0667 decremented by 1 when frog is moved
@066C if ==7, then frog_on_lily_log = joystick_udlr = 0, check collision status etc?????
@09FF (frog_leaping_sound) controls sound

1F9C **hole_control**

7	6	5	4	3	2	1	0
Hole 1	Hole 2	Hole 3	Hole 4		Counter / index (count of number of holes unfilled)		

@028A initialised to $04
@03B3 (start_fresh_run) bits210 predecremented and used as index to LUT_time_allowed[x]
which is then saved to unclear
@047a (subr_init_4_variables; puts frogs-in-holes offscreen) set to $04
@0554 (safely_home) or'ed with byte indicating hole landed in, then decremented and.....
@0560 .....saved

1F9D **max_HC_river**

@028A initialised to $BF
@074E if array_ HC_A_logs[N] >= max_HC_river, array_ HC_A_logs[N] = 0 and start new log shape
@0787 if array_HC_A_lillies[N] ==0, array_ HC_A_lillies[r2] = max_HC_river, and status_A_lillies= 3
@0795 if array_HC_B_lillies[N] >= max_HC_river, array_HC_B_lillies[N] = 0, and status_B_lillies = 3
@07E0 if array_ HC_B_logs[N] ==0, array_ HC_B_logs[N] = max_HC_river
@0BE4 (sub_level_control) if bit 2 of LUT_level_parameters[x]==1, $10 added to it

1F9E **max_HC_road** the right-hand start and stop point for road traffic

@028A initialised to $FF
@076D if array_HC_B_trucks[N] = 0, array_HC_B_trucks[N] = max_HC_road
@07ED if array_HC_A_cars[N] = 0, array_HC_A_cars[N] = max_HC_road
@07F8 if array_HC_A_trucks[N] >= max_HC_road , array_HC_A_trucks[N] = 0
@0810 if array_HC_B_cars[N] >= max_HC_road , array_HC_B_cars[N] = 0
@0BE8 (sub_level_control) if LUT_level_parameters[x]{bit 2} == 1, $10 subtracted from it

1F9F **log_shape** defines length of all logs for current level

@028A initialised to $FF
@071A (log appearing from left) if array_shape_A_logs[N] < log_shape, shift
array_shape_A_logs[N] right and set bit 7
@07DD (log reappears on right) array_shape_B_logs[N] = log_shape
@0BD5 (sub_level_control) if bit 3 of LUT_level_parameters[x]==1, rotated left and bit 0 cleared

1FA0 **level**

@028A initialised to $00
@0445 decremented
@0512 (subr_flash_lily_decision) compared to bits3210 of object1HC
@0B8F (sub_level_control) if level >= $0F, EXIT via 0BF0 (see below)
@0B97 (sub_level_control) used as index to LUT_level_parameters
@0B9C (sub_level_control) level ++
@0BF0 (sub_level_control) level ++, timer_P_reset_value --

@0C12 (subr_select_start) level ++
@0C1F (subr_select_start) level -- , used as decrementing counter during convrsion to BCD
@0C23 (subr_select_start) level = BCD value of binary level
@0C49 (subr_select_start) output to scoreL the level reached

1FA1 **timer_P_reset_value**

@028A initialised to $C8 (200; 4 second timer)
@03BE (start_fresh_run) timer_P = timer_P_reset_value
@09D7 timer_P = timer_P_reset_value
@0BF8 (sub_level_control) when level >= $0F , timer_P_reset_value --

1FA2 **speed_control_counter**

initialised @028A to $10
@06FA (start_screen_output) decremented and if == 0, set to $10

@0701 used as an index to table LUT_speed_control

1FA3 control_bits_2

	7	6	5	4	3	2	1	0
0	1 up			@start line	Player 1	@start line	@start line	
1	2 up	Just landed in hole	Sound fx lost life	Sound fx home	Player 2	(lose a life?)	Start new run	Game beaten

@028A initialised to $80

Bit 7 initialised at start_play according to 1/2UP

Bit 6 @0583, (frog moved to start position) **set** bit 6

@05F0, (bonus_life_and_score) **set** bit 6 when frog landed in hole in bank

@06F3, (start_screen_output) if bit 6 == 1, goto 0711....

@0711 (start_screen_output) ..... clear bit 6

Bit 2 set in frog_mishap_2, cleared once the frog is back at the start line; tested @399 ==1 go to loose life

@0298, (start_play) control_bits_2 = $00; control_bits_2{bit**7** } = control_bits{bit0}

@02B6, (start_new_frame) bit **3** is used to select a/d pot 1 or 2

@0399, (EndOfRun) if bit **2** == 1 , goto player_loses_a_life

@039E, (EndOfRun) if bit **1** = 1 then prepare_for_new_run, else **clear** bit **5** and start_fresh_run

@03EA, (start_fresh_run) bits **421** are **cleared** (frog is now ready at the start line)

@03F5, (player_loses_a_life) bit **3** is tested to determine which player lost a life

@040C, (player 1 dead) bit **7** is tested; game over if player 1 dead and 1UP game

@0419, (switch_player) bit **3** is **inverted** to hand over to player 2

@04A1, (frog_mishap_2) bit **5** and bit **2** are **set**

@0583, (frog moved to start position) **set** bits **64** frog moved to start position

@05BB, (safely_home) if bit **1** == 0, goto set_50_frame_delay

@05BF, (safely_home) if bit **7** == 0, goto set_next_level

@05C6, (safely_home) if bit **3** == 1, goto two_up

@05DD, (safely_home) bit **3** is **inverted**

@05F0, (bonus_life_and_score) **set** bits **641** (hole in bank)

@05F6, (bonus_life_and_score) if bit **3** == 1, goto bonus_player_2

@06F3, (start_screen_output) if bit**6** == 1, goto 0711....

@0711 (start_screen_output) ..... clear bit **6**

@09E0 (control sound effects) bit **0** is tested, if==1 initiates game beaten routine

@09E5 (control sound effects) if bit **4** == 1, change the pitch

@09F0 (control sound effects) if bit **5** == 1, change the pitch

@0CFB, (subr_1or2_player) bit **7** is tested, sets object1HCB to either $20 or **HC_down_arrow**

@0DD7, (subr_increase_score) bit3 tested to see whose score to use
(0=score1, 1=score 2)
@0DFD, (subr_increase_score) **set** bit 0 , GAME HAS BEEN BEATEN

1FA4 **score1L**
initialised @028A to $00
@0C79 passed to subr_display_score

1FA5 **score1H**
initialised @028A to $00
@0C79 passed to subr_display_score

1FA6 **score2L**
initialised @028A to $00
@0C79 passed to subr_display_score

1FA7 **score2H**
initialised @028A to $00
@0C79 passed to subr_display_score

START 5 BYTES HORIZONTAL BAR DEFINITIONS

1FA8 **grid_h1** Not a variable.
This is the horizontal extension control byte for the first six vertical bars in the grid, used in game play to draw the holes in the bank
@0E43 initialised to $0E

1FAC **control_bits**

	7	6	5	4	3	2	1	0
0	1 up					[1]	[2]	1 up
1	2 up	Attract mode					[2]	2 up
	*		*	*	*	#	*	

@01B0 set to $40 at transition from title to play
@0295 (start_play) if bit**0** == 1, then control_bits_2 = $80, else = 0
@02FA bit **6** tested, force 'joystick up' for attract mode
@0382 if bit **2** == 0, goto branch038e [1], (
else clear clear bits 754321 and goto SELECT (*#)
@044F (game_over) clear bits **75431**, set bit **2** (*#)
@0C35 if bit **6** == 1, then timer_Q= 5, else =$19

@0CA9 (subr_select_game) bit **0** inverted when select button released

@0CBE (subr_select_game) clear bits **765321** %000x000x
@0CE5 (subr_1or2_player) bit **0** tested 0 = a one player game, 1= a two player game
@0CFE (subr_1or2_player) bit **7** tested used to decide position of down arrow

[1] branch038e blanks the score display then tests/modifies control_bits_2