

Rogueima I MPV-4

- rogueima.asm: 390
- draw.asm: 203
- map.asm: 53
- mon.asm: 99
- msg.asm: 166
- obj.asm: 167
- talk.asm: 166

Total: 1,242 SLOC. According to SLOCCOUNT, a program by David A. Wheeler, using traditional software development metrics this program would have taken four months and cost \$33,921 to develop. The four months is about right, then again I did this in about 2 months; and frankly, I would have done it for half the cost!

rogueima.sda

```
// Rogueima (C) 2026 Appledog Hu
// rogueima.sda
// rogueima 'main'

; Variables
.equ PX $00      ; player X
.equ PY $01      ; player Y
.equ RX $02      ; robot X
.equ RY $03      ; robot Y

.equ player_str   $04      ; 2 bytes
.equ player_hp    $06      ; 2 bytes
.equ player_name  $08      ; 16 bytes + 1 zero starting at $02
.equ pname_zero   $18      ; this must always be a zero even if there are
earlier zeroes.
.equ game_time    $19      ; 2 bytes
.equ player_score $1b      ; 2 bytes

.equ VIDEO_MODE   $01EF00  ; Current video mode (1 byte)

.address $C000

    ; BASIC stub: 10 SYS 269
    ;.bytes $FB, $0A, $00
    ;.bytes "SYS 269", $00
    ;.bytes $00, $00

; --- Entry point at $00010D (decimal 269) ---
start:
    ; Set mode 6.
```

```
LDAA $40      ; Set video mode
LDAL #6       ; to 6
INT 0x10

CALL @title_screen
CALL @init_game

CALL @get_player_name
LDA #0
STAL [@pname_zero]      ; ensure player name length

JMP @main_loop      ; start game
```

```
init_game:
    ; Initialize variables
    LDAL #19
    STAL [@PX]
    LDAL #11
    STAL [@PY]
    LDAL #40
    STAL [@RX]
    LDAL #22
    STAL [@RY]

    LDA #10
    STA [@player_str]
    STA [@player_hp]

    LDA #0
    STA [@game_time]
    STA [@player_score]

    CALL @init_objects      ; Initialize the game objects list.

    RET
```

```
;;;;;;;;;;;;;;
;; Main Loop
;;;;;;;;;;;;;;
main_loop:
    CALL @draw_map
    CALL @get_input
    CALL @move_mobs
    ;CALL @robot_beep
    CALL @inc_game_time

    JMP @main_loop ; Player can press Q to quit.
```

```
inc_game_time:
    LDA [@game_time]
    INC A
```

```

    STA [@game_time]
    RET

get_player_name:
    LDELM @what_is_your_name
    LDAH $18      ; write string
    INT 0x10

    LDAH $68      ; IO_INPUT
    INT 0x05
    ; player name is now in a system buffer (pointed too by ELM).

    ; Ensure player name is no longer than 16 characters.
    MOV FLD, ELM
    ADD FLD, #17
    LDAL #0
    STAL [FLD]

    ; Copy player name into variable space.
    LDFLD @player_name
name_copy_loop:
    LDAL [ELM, +]
    STAL [FLD, +]
    JNZ @name_copy_loop

    RET ;; return.

what_is_your_name:
    .bytes " What is your name? ", 0

;;;;;;;;;;;;;
;; Get input from the player.
;;;;;;;;;;;;;
get_input:
    ; Blocking GETKEY -> AL
    LDAH $02
    INT $10

    ;; Add the player's key as entropy.
    LDAH $02
    INT 0x13

    ; Uppercase: if AL >= 'a' and AL < 'z'+1, subtract 32
    CMP AL, #97
    JNC @gi_check_keys      ; AL < 'a', skip
    CMP AL, #123
    JC @gi_check_keys       ; AL >= '{', skip
    SUB AL, #32

gi_check_keys:

```

```
CMP AL, #'H'
JZ @move_left
CMP AL, #'J'
JZ @move_down
CMP AL, #'K'
JZ @move_up
CMP AL, #'L'
JZ @move_right

CMP AL, #128
JZ @move_left
CMP AL, #129
JZ @move_down
CMP AL, #130
JZ @move_up
CMP AL, #131
JZ @move_right

CMP AL, #'M'
JZ @spawn_monster
CMP AL, #'T'
JZ @do_talk
CMP AL, #'O'
JZ @open_door
CMP AL, #'G'
JZ @create_gold
CMP AL, #'Q'
JZ @quit_game
CMP AL, #','          // pick things up using comma
JZ @pick_up_items
```

```
RET
```

```
move_left:
```

```
LDAL [@PX]
JZ @ml_done
DEC AL          ; AL = new PX
LDX #0
LDY #0
MOV XL, AL      ; XL = new PX (survives CALL)
LDYL [@PY]
CALL @is_walkable
JNC @ml_done
STXL [@PX]
```

```
ml_done:
```

```
RET
```

```
move_right:
```

```
LDAL [@PX]
INC AL          ; AL = new PX
LDB [@map1_dim] ; BL = map_width
```

```

    CMP AL, BL
    JC @mr_done           ; AL >= map_width, out of bounds, abort
    LDX #0
    LDY #0
    MOV XL, AL            ; XL = new PX
    LDYL [@PY]
    CALL @is_walkable
    JNC @mr_done
    STXL [@PX]
mr_done:
    RET

move_up:
    LDAL [@PY]
    JZ @mu_done
    DEC AL                ; AL = new PY
    LDX #0
    LDY #0
    LDXL [@PX]
    MOV YL, AL            ; YL = new PY (survives CALL)
    CALL @is_walkable
    JNC @mu_done
    STYL [@PY]
mu_done:
    RET

move_down:
    LDAL [@PY]
    INC AL                ; AL = new PY
    LDB [@map1_dim]       ; BH = map_height
    CMP AL, BH
    JC @md_done           ; AL >= map_height, out of bounds, abort
    LDX #0
    LDY #0
    LDXL [@PX]
    MOV YL, AL            ; YL = new PY
    CALL @is_walkable
    JNC @md_done
    STYL [@PY]
md_done:
    RET

;;;;;;;;;;;;;
;; Move the robot.
;;;;;;;;;;;;;
move_mobs:
    RET

;;;;;;;;;;;;;
;; Game Over.
;;;;;;;;;;;;;

```

```
game_over:
    ; CLS
    LDAH $10
    INT $10

    ; Print "OH NO! THE ROBOT CATCHES YOU."
    LDBLX @msg_game_over1
    LDAH $66
    INT $05
    LDAH $64          ; newline
    INT $05

    ; Print "GAME OVER."
    LDBLX @msg_game_over2
    LDAH $66
    INT $05
    LDAH $64
    INT $05

    RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Player quit game
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
quit_game:
    ; CLS
    LDAH $10
    INT $10

    ; Print "QUIT GAME"
    LDBLX @msg_quit
    LDAH $66
    INT $05

    POP ELM ; destroy return address of CALL from main loop
    RET     ; exit program.

robot_beep:
    LDELM @robot_sfx
    LDAH $52
    INT $11
    RET

;
=====
; String data
;
=====
msg_game_over1:
    .bytes "OH NO! THE ROBOT CATCHES YOU.", 0
msg_game_over2:
```

```

    .bytes "GAME OVER.", 0
msg_quit:
    .bytes "QUIT GAME", 10,13,0
robot_sfx:
    .bytes "T120 W1 V5 02 L24 B", 0

; =====
; Title Screen
; =====
title_screen:
    ; Clear screen
    ; (This is done on mode switch, but we do it here anyways.)
    LDAH $10
    INT $10

    LDA $1500          ; Set cursor
    LDX #1
    LDY #1
    INT 0x10

    LDELM @str_title1
    LDA $1800
    INT 0x10
    RET

str_title1:
    .bytes "Rogueima I: The First Dungeon",10,13
    .bytes " Copyright 2026 by Appledog Hu.",10,10,13,0

;;;;;;;;;;;;;
;; get_glyph(X, Y) -> AL
;;
;; Returns the map glyph at column X, row Y.
;; Caller is responsible for X, Y being within map bounds.
;;
;; Inputs:   X = column, Y = row
;; Output:   AL = glyph byte (e.g. '#', '.', '+', ' ')
;;
get_glyph:
    PUSH ELM
    PUSH FLD
    PUSH K
    PUSH Z
    PUSH I
    PUSH B

    ; Default: map tile glyph
    LDK [@map1_dim]
    MOV Z, Y

```

```
MUL Z, KL
ADD Z, X
LDELM @map1_data
ADD ELM, Z
LDAL [ELM] ; AL = map glyph (default return)

; Override if any object sits at (X, Y)
LDFLD @OBSJS_BASE
get_glyph_obj_loop:
LDFLD [FLD]
CMP FLD, @OBSJS_BASE
JZ @get_glyph_done ; wrapped to head, no match, keep map glyph

LDI @OBJ_ID
LDB [FLD+I]
JZ @get_glyph_obj_loop ; uninitialized slot

LDI @OBJ_X
LDBL [FLD+I]
CMP BL, XL
JNZ @get_glyph_obj_loop

LDI @OBJ_Y
LDBL [FLD+I]
CMP BL, YL
JNZ @get_glyph_obj_loop

LDI @OBJ_VIS
LDAL [FLD+I]

get_glyph_done:
POP B
POP I
POP Z
POP K
POP FLD
POP ELM
RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; put_glyph(X, Y, AL)
;;
;; Places the glyph AL at x, y
;; Caller is responsible for X, Y being within map bounds.
;;
;; Inputs: X = column, Y = row, AL = chr (ex. 'A')
;;
put_glyph:
PUSH ELM
PUSH K
PUSH Z
```



```

    LDK [@map1_dim]      ; KL = map_width
    MOV Z, Y             ; Z = Y
    MUL Z, KL            ; Z = Y * map_width
    ADD Z, X             ; Z = Y * map_width + X
    LDELM @map1_data
    ADD ELM, Z           ; ELM -> map[Y][X]
    STAL [ELM]          ; AL = glyph

    POP Z
    POP K
    POP ELM
    RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; is_walkable(X, Y)
;;
;; Sets CARRY if this tile is walkable.
;;
;; Non-walkable tiles: = and +
;;

is_walkable:
    PUSHA
    CALL @get_glyph      ; AL now has glyph.
    CMP AL, #'#'
    JZ @is_walkable_no
    CMP AL, #'+'
    JZ @is_walkable_no

    ;; default: yes.
is_walkable_yes:
    POPA
    SEC
    RET

is_walkable_no:
    POPA
    CLC
    RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; open_door()
;; This replicates the gi_get_keys pattern
;; but is used to directionally change a door's status.
;;
open_door:
    ; Show "Open where?"
    LDELM @str_open_where
    CALL @print_msg
    CALL @msg_display

```

```
; Ask player for key
LDAH $02      ; Blocking GETKEY -> AL
INT $10

;; Add the player's key as entropy.
LDAH $02
INT 0x13

; Uppercase: if AL >= 'a' and AL < 'z'+1, subtract 32
CMP AL, #97
JNC @od_check_keys      ; AL < 'a', skip
CMP AL, #123
JC @od_check_keys       ; AL >= '{', skip
SUB AL, #32
```

```
od_check_keys:
; Check direction given
CMP AL, #'H'
JZ @open_door_left
CMP AL, #'J'
JZ @open_door_down
CMP AL, #'K'
JZ @open_door_up
CMP AL, #'L'
JZ @open_door_right

CMP AL, #128
JZ @open_door_left
CMP AL, #129
JZ @open_door_down
CMP AL, #130
JZ @open_door_up
CMP AL, #131
JZ @open_door_right

; Bad direction -- ignore.

RET
```

```
open_door_up:
; Show "open door north"
LDELM @str_OPEN_DOOR
CALL @print_msg
LDELM @str_NORTH
CALL @print_msg
LDELM @str_crlf
CALL @print_msg

LDI #0
LDJ #0
LDK #0
```

```
LDL #1
JMP @open_door_go
```

```
open_door_down:
LDELM @str_OPEN_DOOR
CALL @print_msg
LDELM @str_SOUTH
CALL @print_msg
LDELM @str_crlf
CALL @print_msg
```

```
LDI #0
LDJ #1
LDK #0
LDL #0
JMP @open_door_go
```

```
open_door_left:
LDELM @str_OPEN_DOOR
CALL @print_msg
LDELM @str_WEST
CALL @print_msg
LDELM @str_crlf
CALL @print_msg
```

```
LDI #0
LDJ #0
LDK #1
LDL #0
JMP @open_door_go
```

```
open_door_right:
LDELM @str_OPEN_DOOR
CALL @print_msg
LDELM @str_EAST
CALL @print_msg
LDELM @str_crlf
CALL @print_msg
```

```
LDI #1
LDJ #0
LDK #0
LDL #0
JMP @open_door_go
```

```
open_door_go:
LDX #0
LDY #0
LDXL [@PX]
LDYL [@PY]
ADD XL, IL
```

```
ADD YL, JL
SUB XL, KL
SUB YL, LL
CALL @get_glyph
CMP AL, #'+'
JZ @open_door_open_it
CMP AL, #'-'
JZ @open_door_close_it

;; bad command? ignore.
RET
```

```
open_door_open_it:
    LDAL #'-'
    CALL @put_glyph
    RET
```

```
open_door_close_it:
    LDAL #'+'
    CALL @put_glyph
    RET
```

draw.sda

```
// rogueima (C) 2026 Appledog Hu
// draw.asm
// Draw the display screen; borders, status, chat, wold map, etc.

cam_origin_x:
    .bytes 0,0          ; displacement value for drawing on the projected map

cam_origin_y:
    .bytes 0,0          ; displacement value for drawing on the projected map

draw_map:
    LDAH $51            ; VSTOP
    INT 0x18

    CALL @draw_borders
    CALL @draw_world
    CALL @draw_items
    CALL @draw_player
    CALL @draw_mobs
    CALL @draw_stats
    CALL @draw_msgs
    CALL @msg_display

    LDAH $50            ; VSTART
```

```
    INT 0x18

    RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Draw borders.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
draw_borders:
    ; CLS
    LDAH $10          ; CLS
    INT $10

    LDX #0    ;; clear XH and YH.
    LDY #0

    ; Draw top/bottom border: '*' at (0..79, 0) and (0..79, 24)
    LDBL #'*'
    LDXL #0

dm_top_bottom:
    LDYL #0
    LDAH $11          ; write char AL at XL, YL
    INT $10

    LDYL #24
    LDAH $11          ; write char
    INT $10

    CMP XL, #59        ; if XL >= 10, then set carry.
    JNC @dm_skip_ms    ; So if it's not, then don't draw the middle
separator char.

    LDYL #10          ; status/text window separator
    LDAH $11          ; write char
    INT $10

dm_skip_ms:
    INC XL
    CMP XL, #80        ; Mode 6 is 80 chars wide.
    JNC @dm_top_bottom

    ; Draw left/right borders: '*' at (0, 0..24) and (59, 0..24) and (79,
0..24)
    LDBL #'*'
    LDYL #0
dm_left_right:
    LDXL #0          ; left border
    LDAH $11          ; write char
    INT $10

    LDXL #59          ; middle border
```

```
LDAH $11          ; write char
INT $10

LDXL #79          ; right side border
LDAH $11          ; write char
INT $10

INC YL
CMP YL, #25
JNC @dm_left_right
RET
```

draw_player:

```
LDBL #'@'
LDXL [@PX]
LDYL [@PY]
LDIL [@cam_origin_x]
LDJL [@cam_origin_y]
SUB XL, IL          ; XL = PX - I_start
ADD XL, #1          ; play_x_offset (mirror draw_world)
SUB YL, JL          ; YL = PY - J_start
ADD YL, #1          ; play_y_offset
LDAH $11
INT $10
RET
```

draw_mobs:

```
; stub for now
RET
```

draw_stats:

```
; Set cursor
LDA $1500
LDX #61
LDY #2
INT 0x10
```

```
LDELM @str_name    ; Draw 'Name: '
LDA $1800           ; write string
INT 0x10
```

```
; Draw player's name.
; This will draw right after the 'Name: ' above.
; That's why we added spaces to the strings (below).
LDELM @player_name
LDA $1800           ; write string
INT 0x10
```

```
;;;;;;;;;;;;;; Strength
LDA $1500           ; set cursor
LDX #61
```

```
LDY #4
INT 0x10

LDELM @str_str      ; Draw 'Str: '
LDA $1800           ; write string
INT 0x10

; Draw player's strength score
LDA $6300           ; print unsigned word (will print after string
above).
LDB [@player_str]
INT 0x05

;;;;;;;;;;;;;; Health
LDA $1500           ; set cursor
LDX #61
LDY #5
INT 0x10

LDELM @str_hp       ; Draw 'Health: '
LDA $1800           ; Write string
INT 0x10

LDA $6300           ; Write number (unsigned)
LDB [@player_hp]
INT 0x05

;;;;;;;;;;;;;; Score
LDA $1500           ; Set cursor
LDX #61
LDY #7
INT 0x10

LDELM @str_score    ; Draw 'Score: '
LDA $1800           ; write string
INT 0x10

LDA $6300           ; print number
LDB [@player_score]
INT 0x05

;;;;;;;;;;;;;; Time
LDA $1500           ; set cursor
LDX #61
LDY #8
INT 0x10

LDELM @str_time     ; Draw 'Time: '
LDA $1800           ; write string
```

```
    INT 0x10

    LDA $6300          ; print number
    LDB [@game_time]
    INT 0x05

    RET

str_name:
    .bytes "Name: ", 0

str_str:
    .bytes "STR: ", 0

str_hp:
    .bytes "HP: ", 0

str_time:
    .bytes "T: ", 0

str_score:
    .bytes "Score: ", 0

draw_msgs:
    ; We will work on this after.
    RET

draw_world:
    LDT [@disp_dim]    ; TL = play_width,  TH = play_height
    LDK [@map1_dim]    ; KL = map_width,    KH = map_height

    LDX #0
    LDY #0
    LDXL [@PX]
    LDYL [@PY]
    MOV I, X
    MOV J, Y

    ; Clamp I to [0, KL - TL]
    MOV AL, TL
    DEC AL
    SHR AL, #1          ; AL = (TL-1)/2 = centered offset
    CMP IL, AL
    JC @dw_i_lo_ok
    MOV IL, AL
dw_i_lo_ok:
    SUB IL, AL

    MOV AL, KL
    SUB AL, TL          ; AL = max_I_start
```



```
    CMP IL, AL
    JNC @dw_i_hi_ok
    MOV IL, AL
dw_i_hi_ok:

    ; Clamp J to [0, KH - TH]
    MOV AL, TH
    DEC AL
    SHR AL, #1
    CMP JL, AL
    JC @dw_j_lo_ok
    MOV JL, AL
dw_j_lo_ok:
    SUB JL, AL

    MOV AL, KH
    SUB AL, TH
    CMP JL, AL
    JNC @dw_j_hi_ok
    MOV JL, AL

dw_j_hi_ok:
    STIL [@cam_origin_x]      ; Save computed camera origin
    STJL [@cam_origin_y]

    ; Render loop
    LDX #1
    LDY #1
    MOV G, I

dw_x_loop:
    LDELM @map1_data
    ADD ELM, I
    MOV Z, J
    MUL Z, KL
    ADD ELM, Z
    LDBL [ELM]

    LDAH $11
    INT 0x10

    INC I
    INC X
    CMP TL, XL                ; continue while TL >= X
    JC @dw_x_loop

    LDX #1
    MOV I, G
    INC J
    INC Y
    CMP TH, YL
```

```
JC @dw_x_loop      ; same trick on the outer
```

```
RET
```

```
disp_dim:
    .bytes #58, #23
```

map.sda

```
;;;;;;;;;;;;;;
;; map data
;;
map1_id:
    .bytes 1
map1_name:
    .bytes "START  ", 0      ; 8 characters and zero terminated
map1_up:
    .bytes 5, 5
map1_down:
    .bytes 25, 15
map1_dim:
    .bytes 80, 40
map1_data:
    .bytes
    "#####
#####" ; 0
    .bytes "#          #"
#" ; 1
    .bytes "#          #"
#" ; 2
    .bytes "#          #"
#" ; 3
    .bytes "#          #"
#" ; 4
    .bytes "#          +"
#" ; 5
    .bytes "#          #"
#" ; 6
    .bytes "#          #          #####
#" ; 7
    .bytes "#          #          #
#" ; 8
    .bytes "#          #          #          +
#" ; 9
    .bytes "#####+#####          #          #
#" ; 10
    .bytes "#          #          #          #
#" ; 11
```

```

    .bytes "#                      #####
#" ; 12
    .bytes "#
#" ; 13
    .bytes "#
#" ; 14
    .bytes "#
#" ; 15
    .bytes "#
#" ; 16
    .bytes "#
#" ; 17
    .bytes "#
#" ; 18
    .bytes "#
#" ; 19
    .bytes "#
#" ; 20
    .bytes "#
#" ; 21
    .bytes "#
#" ; 22
    .bytes "#
#" ; 23
    .bytes "#
#" ; 24
    .bytes "#
#" ; 25
    .bytes "#
#" ; 26
    .bytes "#
#" ; 27
    .bytes "#
#" ; 28
    .bytes "#
#" ; 29
    .bytes "#
#" ; 30
    .bytes "#
#####+#####+#####+#####" ; 31
    .bytes "#                      #          #
#          #" ; 32
    .bytes "#                      #          #
#          #" ; 33
    .bytes "#                      #          #
#          #" ; 34
    .bytes "#                      #          #
#          #" ; 35
    .bytes "#                      #          #
#          #" ; 36
    .bytes "#                      #          #

```

```
#           #" ; 37
    .bytes "#           #           #
#           #" ; 38
    .bytes
"#####
#####" ; 39
```

mon.sda

```
;; mon.sda
;; spawning monsters with the m command (as a test of the system, of
course!)

spawn_monster:
    ; Find a free space in the object list
    LDA #0
    LDELM [@OBJ_S_BASE] ; start scanning at HEAD.FLINK (i.e. the first
object)
    LDI @OBJ_ID
    SCANQUE ELM, A, I
    JNZ @spawn_monster_oom

    ; Since we're using a sentinel node, if the sentinel is a match it means
there was no valid node.
    CMP ELM, @OBJ_S_BASE
    JZ @spawn_monster_oom

    LDA [@obj_ids]
    LDI @OBJ_ID
    STA [ELM+I]
    INC A
    STA [@obj_ids]

    LDA #2 ; obj type 1 = gold
    LDI @OBJ_TYPE
    STA [ELM+I]

    CALL @random_monster ; set up a random monster

    ; get a random x,y location that is walkable
    ; this could be refactored out maybe
    ; it's in create_gold too
    LDK [@map1_dim]
spawn_mon_xy:
    LDAH $00 ; 16 bit random number
    INT 0x13
    MOD B, KL
    MOV X, B
```

```
LDAH $00      ; 16 bit random number
INT 0x13
MOD B, KH
MOV Y, B
CALL @is_walkable
JNC @spawn_mon_xy
```

```
LDI @OBJ_X
STX [ELM+I]
LDI @OBJ_Y
STY [ELM+I]
```

```
CLC           ; no error
RET           ; return
```

spawn_monster_oom:

```
SEC           ; set carry on error
RET
```

random_monster:

```
;; What kind of monster do we want?
;; Pick a random number from 1 to 3.
```

randmon_montype:

```
LDAH $00      ; 16 bit random number
INT 0x13      ; math services
MOD B, #3
INC B         ; amount is now 1 to 3.
CMP B, #1
JZ @make_rat
CMP B, #2
JZ @make_snake
CMP B, #3
JZ @make_spider
```

```
;; it has to be one of the above 3.
;; But for safety..
JMP @randmon_montype
```

make_rat:

```
;; Monster Hit Points
LDAH $00      ; 16 bit random number
INT 0x13      ; math services
MOD B, #2
INC B         ; amount is now 1 or 2 for a rat.
LDI @OBJ_DATA1
STB [ELM+I]
```

```
;; Monster strength (does how much damage)
LDB #1
LDI @OBJ_DATA2
STB [ELM+I]
```

```
; Add GLYPH and VIS.  
LDAL #'r'          ; r for rat  
LDI @OBJ_GLYPH  
STAL [ELM+I]  
LDI @OBJ_VIS  
STAL [ELM+I]  
RET
```

make_snake:

```
;; Monster Hit Points  
LDAH $00          ; 16 bit random number  
INT 0x13          ; math services  
MOD B, #3  
INC B             ; amount is now 1 to 3 for a snake.  
LDI @OBJ_DATA1  
STB [ELM+I]
```

```
;; Monster strength (does how much damage)  
LDB #2  
LDI @OBJ_DATA2  
STB [ELM+I]
```

```
; Add GLYPH and VIS.  
LDAL #'s'          ; s for snake  
LDI @OBJ_GLYPH  
STAL [ELM+I]  
LDI @OBJ_VIS  
STAL [ELM+I]  
RET
```

make_spider:

```
;; Monster Hit Points  
LDAH $00          ; 16 bit random number  
INT 0x13          ; math services  
MOD B, #5  
INC B             ; amount is now 1 to 5 for a big fat spider  
LDI @OBJ_DATA1  
STB [ELM+I]
```

```
;; Monster strength (does how much damage)  
LDB #3  
LDI @OBJ_DATA2  
STB [ELM+I]
```

```
; Add GLYPH and VIS.  
LDAL #'x'          ; x for bugs, insects, spiders, etc.  
LDI @OBJ_GLYPH  
STAL [ELM+I]  
LDI @OBJ_VIS  
STAL [ELM+I]  
RET
```

msg.sda

```
; msg.sda -- manage the message window.

msg_dim_x:
    .bytes #19, #0
msg_dim_y:
    .bytes #13, #0    ; width and height

msg_home_x:
    .bytes #60, #0
msg_home_y:
    .bytes #11, #0    ; X and Y where to start drawing

msg_cursor_x:
    .bytes #0, #0      ; X internal cursor location (start: left side)
msg_cursor_y:
    .bytes #12, #0     ; Y internal cursor location (start: lower row)

msg_data:
    ; 13 rows of 19 bytes (must match msg_dim)
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    .bytes 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; print_msg(ELM)
;;
;; ELM points to a string. This function
;; simplifies a "puts" type function by
;; calling the "putc" type function.

print_msg:
    PUSH A                ; We over-write AL so save register A.
    PUSH ELM              ; Save pointer to string

print_msg_loop:
    LDAL [ELM, +]         ; Load char and advance ELM.
```

```
JZ @print_msg_done      ; If we hit a zero, (end of string), then exit.
CALL @msg_putchar      ; Writes char al at position in msg_cursor
JMP @print_msg_loop

print_msg_done:
    POP ELM              ; restore original string pointer
    POP A
    RET

;;;;;;;;;;;;;;;;;;;;;;;;;;
;; msg_putchar(AL)
;; looks at msg_cursor, msg_dim, etc. and writes the character.
;;
msg_putchar:
    ;; We are called with character in AL.
    ;; Find the current cursor position:
    LDX [@msg_cursor_x]
    LDY [@msg_cursor_y]

    ; Is AL a special character?
    CMP AL, #10          ; line feed
    JZ @msg_putchar_lf
    CMP AL, #13          ; carriage return
    JZ @msg_putchar_cr

mpc_check_done:
    ; Not a special character. Print it.
    LDFLD @msg_data
    ADD FLD, X
    LDI [@msg_dim_x]
    MUL Y, I             ; multiply Y by row length to get ptr
    ADD FLD, Y
    STAL [FLD]           ; Store this character in the buffer.

    ; Advance the cursor properly.
    CALL @msg_putchar_advance
    LDX [@msg_cursor_x]

msg_putchar_done:
    RET

msg_putchar_lf:
    CALL @msg_do_lf
    JMP @msg_putchar_done

msg_putchar_cr:
    CALL @msg_do_cr
    JMP @msg_putchar_done

;;;;;;;;;;;;;;;;;;;;;;;;;;
```



```
;; Advance the cursor properly.
;; This is a big responsibility!
;; We may need to linefeed the msg window.
;;
msg_putchar_advance:
    PUSHA
    LDX [@msg_cursor_x]
    LDY [@msg_cursor_y]

    INC X      ; Attempt to move cursor.

    ;; Check if we moved past the X-width
    LDI [@msg_dim_x]
    CMP X, I    ; Since X is 0 based, if it is equal to I it's out of
bounds.
    JNZ @mpc_done ; X is fine, so we're done.

    ; We've moved past the right side, do a CR/LF.
    CALL @msg_do_cr      ; Carriage return
    CALL @msg_do_lf      ; Do a linefeed, will scroll if necessary.

mpc_done:
    STX [@msg_cursor_x]
    STY [@msg_cursor_y]
    POPA
    RET

;;;;;;;;;;;;;;
;; msg_do_cr
;; this doesn't affect the buffer in any way, just the cursor.
;;
msg_do_cr:
    LDX #0
    STX [@msg_cursor_x]
    RET

;;;;;;;;;;;;;;
;; msg_do_lf
;; Just lf.
;; Adds a line only if we need to.
msg_do_lf:
    PUSH J
    INC Y
    LDJ [@msg_dim_y]
    CMP Y, J
    JNC @msg_do_lf_done
    CALL @msg_scroll ; manually scroll window.

msg_do_lf_done:
    STY [@msg_cursor_y]
```

```
POP J
RET

;;;;;;;;;;;;;
;; msg_scroll
;; Called by msg_do_lf.
;; You probably don't want to call this by itself.
;; Note that it modifies Y, and the caller may depend on this.
msg_scroll:
    PUSH ELM
    PUSH FLD
    CMP Y, 0                ; if Y is zero,
    JZ @msg_scroll_noddec   ; don't dec Y.

    DEC Y                  ; We keep the cursor at the same position it was at in the
buffer.
    STY [@msg_cursor_y]

msg_scroll_noddec:
    LDI [@msg_dim_x]        ; get row size
    LDJ [@msg_dim_y]        ; get height
    LDELM @msg_data         ; get start of msg area
    LDFLD @msg_data         ; copy start into FLD
    ADD ELM, I              ; skip one row in ELM (start)
    DEC J                   ; don't copy last row (# of bytes)
    MUL J, I                ; find total bytes to move except last row

msg_scroll_loop:
    LDAL [ELM, +]
    STAL [FLD, +]
    DEC J
    CMP J, #0
    JNZ @msg_scroll_loop

    LDFLD @msg_data
    LDI [@msg_dim_x]
    LDJ [@msg_dim_y]
    DEC J                   ; AL = last row index
    MUL J, I                ; AL = byte offset of last row
    ADD FLD, J

    LDAL $20                ; space
msg_scroll_clear:
    STAL [FLD, +]
    DEC I
    JNZ @msg_scroll_clear   ; write a blank line

    POP FLD
    POP ELM
    RET                    ; Buffer has been updated!
```

```

;;;;;;;;;;;;;
;; msg_display
;; Copies the message buffer to the screen at (msg_home_x, msg_home_y).
;; Treats zero bytes as spaces so the all-zeros init looks blank.
;;
msg_display:
    PUSHA
    LDELM @msg_data          ; ELM walks through the buffer

    LDY #0
    LDYL [@msg_home_y]      ; Y = current screen row
    LDJL [@msg_dim_y]       ; JL = rows remaining

mdpy_row:
    LDX #0
    LDXL [@msg_home_x]      ; X = current screen col (start of row)
    LDIL [@msg_dim_x]       ; IL = cols remaining in this row

mdpy_col:
    LDBL [ELM, +]           ; BL = char, advance ELM
    JNZ @mdpy_print
    LDBL #' '               ; map 0 -> space for the renderer
mdpy_print:
    LDAH $11
    INT $10                 ; write char BL at XL, YL

    INC X
    DEC IL
    JNZ @mdpy_col

    INC Y
    DEC JL
    JNZ @mdpy_row

    POPA
    RET

;;;;;;;;;;;;;
;; String data
;;;;;;;;;;;;;
str_open_where:
    .bytes "> Open where?", 13, 10, 0
str_NORTH:
    .bytes "NORTH", 0
str_EAST:
    .bytes "EAST", 0
str_SOUTH:
    .bytes "SOUTH", 0
str_WEST:
    .bytes "WEST", 0

```

```
str_OPEN_DOOR:
    .bytes "  OPEN DOOR ", 0
str_crlf:
    .bytes #13, #10, #0
```

obj.sda

```
; obj.sda
; create and such for game objects (items and monsters).

; Object structure (offset includes 6 byte INSQUE header)
.equ OBJ_FLINK  0          ; forward-link (managed by INSQUE)
.equ OBJ_BLINK  3          ; back-link (managed by INSQUE)
.equ OBJ_ID     6          ; 2 bytes, max 64k objects
.equ OBJ_TYPE   8          ; 2 bytes; ex. 0 = nothing, 1 = wall, 2=gold, etc.
.equ OBJ_X      10         ; 2 bytes; x position on map
.equ OBJ_Y      12         ; 2 bytes; x position on map
.equ OBJ_GLYPH  14         ; 1 byte; what it looks like (char)
.equ OBJ_VIS    15         ; 1 byte; what it looks like on the map (char)
.equ OBJ_NAME   16         ; 16 bytes + 1 zero (16 max len str)
.equ OBJ_DATA1  33         ; 2 bytes (data 1, ex. gold value)
.equ OBJ_DATA2  35         ; 2 bytes (data 1, ex. gold value)

.equ OBJJS_BASE $00F000    ; objects data starts here
.equ OBJJS_END  $00FFFF    ; last byte for space
.equ OBJ_LEN    37         ; record length for an OBJ including 6 byte header

objs_ptr:
    .bytes 0, 0, 0          ; PTR to last valid object
objs_max:
    .bytes 0, 0            ; max number of objects possible to have

str_gold:
    .bytes "gold", 0

obj_ids:
    .bytes 1, 0 ; Start at ID = 1 (0 means, not an object).

init_objects:
    ; Write blank HEAD node at @OBJJS_BASE (this clears the whole list)
    LDELM @OBJJS_BASE      ; ELM = node ptr
    LDFLD @OBJJS_BASE      ; FLD = FLINK/BLINK ptr
    STELM [FLD, +]         ; write FLINK and advance to BLINK
    STELM [FLD]            ; write BLINK (and leave FLD pointed at
HEAD.BLINK)

    ADD ELM, #6            ; advance past head to first data node space.
```

```

    LDGLK @OBJJS_END
    SUB GLK, @OBJJ_LEN
    INC GLK                      ; GLK now equals first invalid address

init_obj_loop:
    ; 1. test if there's enough memory for another object
    CMP GLK, ELM                ; Will we overflow memory if we initialize an
object here?
    JNC @init_obj_done          ; Yes, so exit (ELM >= GLK == CF).

    ; 2. initialize this object space
    LDI @OBJ_ID
    STB [ELM + I]                ; Write id = 0 for this record
    INSQUE ELM, [FLD]            ; FLD is a ptr to HEAD.BLINK; Append to list

    ; 3. Record number of objects we've initialized
    LDA [@objjs_max]
    INC A
    STA [@objjs_max]

    ; 4. loop
    ADD ELM, @OBJJ_LEN           ; point to next object
    JMP @init_obj_loop           ; loop

init_obj_done:
    LDELM [@OBJJS_BASE]          ; Load HEAD.FLINK (Point to first object on
list)
    RET

create_gold:
    ; Find a free space in the object list
    LDA #0
    LDELM [@OBJJS_BASE]          ; start scanning at HEAD.FLINK (i.e. the first
object)
    LDI @OBJ_ID
    SCANQUE ELM, A, I
    JNZ @create_gold_oom

    ; Since we're using a sentinel node, if the sentinel is a match it means
there was no valid node.
    CMP ELM, @OBJJS_BASE
    JZ @create_gold_oom

    LDA [@obj_ids]
    LDI @OBJ_ID
    STA [ELM+I]
    INC A
    STA [@obj_ids]

    LDA #1                      ; obj type 1 = gold
    LDI @OBJ_TYPE

```

```
    STA [ELM+I]

    ;; Now that we've saved type, let's get its X, Y and amount data.
    LDAH $00      ; 16 bit random number
    INT 0x13      ; math services
    MOD B, #100
    INC B          ; amount is now 1 to 100.
    LDI @OBJ_DATA1
    STB [ELM+I]

    LDK [@map1_dim]
create_gold_xy:
    LDAH $00      ; 16 bit random number
    INT 0x13
    MOD B, KL
    MOV X, B

    LDAH $00      ; 16 bit random number
    INT 0x13
    MOD B, KH
    MOV Y, B

    CALL @is_walkable
    JNC @create_gold_xy

    ; store final x,y location
    LDI @OBJ_X
    STX [ELM+I]

    LDI @OBJ_Y
    STY [ELM+I]

    ; Add GLYPH and VIS.
    LDAL #'$'
    LDI @OBJ_GLYPH
    STAL [ELM+I]
    LDI @OBJ_VIS
    STAL [ELM+I]

    CLC           ; no error
    RET           ; return

create_gold_oom:
    SEC           ; set carry on error
    RET

;;;;;;;;;;;;;
;; Draw all the items that are placed on the map.
;;
draw_items:
    LDDL [@cam_origin_x]      ; DL = I_start, DH = J_start
```

```
LDDH [@cam_origin_y]          ; DL = I_start, DH = J_start

LDT [@disp_dim]                ; TL, TH for upper-bound additions
LDELM @OBS_BASE
draw_items_loop:
LDELM [ELM]
CMP ELM, @OBS_BASE
JZ @draw_items_done

LDI @OBJ_ID
LDA [ELM+I]
JZ @draw_items_loop

LDI @OBJ_X
LDX [ELM+I]
JZ @draw_items_loop
LDI @OBJ_Y
LDY [ELM+I]
JZ @draw_items_loop

; is (XL, YL) inside the viewport?
CMP XL, DL
JNC @draw_items_loop          ; XL < I_start (off-screen left)
MOV AL, DL
ADD AL, TL
CMP XL, AL
JC @draw_items_loop           ; XL >= I_start + TL (off-screen right)

CMP YL, DH
JNC @draw_items_loop          ; YL < J_start (off-screen up)
MOV AL, DH
ADD AL, TH
CMP YL, AL
JC @draw_items_loop           ; YL >= J_start + TH (off-screen down)

; xlate map to screen
SUB XL, DL
ADD XL, #1                    ; play_x_offset (mirror draw_world's screen_x)
SUB YL, DH
ADD YL, #1                    ; play_y_offset (use whatever draw_world uses)

; draw
LDI @OBJ_VIS
LDBL [ELM+I]
LDAH $11
INT $10
JMP @draw_items_loop
draw_items_done:
RET

;;;;;;;;;;
```

```
;; Pick up items
;; Picks up the first pile of gold where the player is standing.
;;
pick_up_items:
    LDELM @OBJ_S_BASE          ; Start at first object

pick_up_items_loop:
    LDELM [ELM]                ; Traverse to next object
    CMP ELM, @OBJ_S_BASE
    JZ @pick_up_items_done     ; End loop if we returned to HEAD node.

    LDI @OBJ_ID
    LDA [ELM+I]                ; get object ID.
    JZ @pick_up_items_loop     ; Skip non-initialized/empty objects

    ; Does this item match the player's location?
    LDI @OBJ_X
    LDX [ELM+I]                ; object x

    LDIL [@PX]                 ; player x
    CMP IL, XL
    JNZ @pick_up_items_loop    ; No match, keep looking.

    LDI @OBJ_Y
    LDY [ELM+I]                ; objecy y

    LDJL [@PY]                 ; player y
    CMP JL, YL
    JNZ @pick_up_items_loop    ; No match, keep looking.

    ; Oh, there's an item here!
    LDI @OBJ_TYPE
    LDA [ELM+I]
    CMP A, #1                  ; is it gold?
    JNZ @pick_up_items_loop    ; No, keep looking.

    ; How much gold is it?
    LDI @OBJ_DATA1
    LDA [ELM+I]
    LDB [@player_score]
    ADD B, A                    ; Add one point for every gold.
    STB [@player_score]        ; Save score.

    LDA #0
    LDI @OBJ_ID                ; Remove the gold object from the game
    STA [ELM+I]                ; by zeroing it's ID (thats all!)

pick_up_items_done:
    RET
```


talk.sda

```
; Talk.sda
; handle talking.
; Right now you can only talk to a monster, unfortunately.
; Or an item?

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; do_talk()
;; This started as a copy of open_door
;;

do_talk:
    ; Show "Talk where?"
    LDELM @str_talk_where
    CALL @print_msg
    CALL @msg_display

    ; Ask player for key
    LDAH $02      ; Blocking GETKEY -> AL
    INT $10

    ;; Add the player's key as entropy.
    LDAH $02
    INT 0x13

    ; Uppercase: if AL >= 'a' and AL < 'z'+1, subtract 32
    CMP AL, #97
    JNC @talk_check_keys      ; AL < 'a', skip
    CMP AL, #123
    JC @talk_check_keys      ; AL >= '{', skip
    SUB AL, #32

talk_check_keys:
    ; Check direction given
    CMP AL, #'H'
    JZ @talk_west
    CMP AL, #'J'
    JZ @talk_south
    CMP AL, #'K'
    JZ @talk_north
    CMP AL, #'L'
    JZ @talk_east

    CMP AL, #128
    JZ @talk_west
    CMP AL, #129
    JZ @talk_south
    CMP AL, #130
```

```
JZ @talk_north
CMP AL, #131
JZ @talk_east

; Bad direction -- ignore.

RET
```

```
talk_north:
; Show "talk north"
LDELM @str_TALK
CALL @print_msg
LDELM @str_NORTH
CALL @print_msg
LDELM @str_crlf
CALL @print_msg

LDI #0
LDJ #0
LDK #0
LDL #1
JMP @talk_go
```

```
talk_south:
LDELM @str_TALK
CALL @print_msg
LDELM @str_SOUTH
CALL @print_msg
LDELM @str_crlf
CALL @print_msg

LDI #0
LDJ #1
LDK #0
LDL #0
JMP @talk_go
```

```
talk_west:
LDELM @str_TALK
CALL @print_msg
LDELM @str_WEST
CALL @print_msg
LDELM @str_crlf
CALL @print_msg

LDI #0
LDJ #0
LDK #1
LDL #0
JMP @talk_go
```

```
talk_east:
    LDELM @str_TALK
    CALL @print_msg
    LDELM @str_EAST
    CALL @print_msg
    LDELM @str_crlf
    CALL @print_msg

    LDI #1
    LDJ #0
    LDK #0
    LDL #0
    JMP @talk_go

talk_go:
    LDX #0
    LDY #0
    LDXL [@PX]
    LDYL [@PY]
    ADD XL, IL
    ADD YL, JL
    SUB XL, KL
    SUB YL, LL
    CALL @get_glyph

    CMP AL, #' '
    JZ @talk_air
    CMP AL, #'#'
    JZ @talk_wall
    CMP AL, #'+'
    JZ @talk_cdoor
    CMP AL, #'-'
    JZ @talk_odoor
    CMP AL, #'r'
    JZ @talk_rat
    CMP AL, #'s'
    JZ @talk_snake
    CMP AL, #'x'
    JZ @talk_spider

    ;; bad command? ignore.
    RET

talk_air:
    LDELM @str_ytt
    CALL @print_msg
    LDELM @str_talk_air
    CALL @print_msg
    RET

talk_wall:
```

```
LDELM @str_ytt
CALL @print_msg
LDELM @str_talk_wall
CALL @print_msg
RET
```

```
talk_cdoor:
  LDELM @str_ytt
  CALL @print_msg
  LDELM @str_talk_cdoor
  CALL @print_msg
  RET
```

```
talk_odoor:
  LDELM @str_ytt
  CALL @print_msg
  LDELM @str_talk_odoor
  CALL @print_msg
  RET
```

```
talk_rat:
  LDELM @str_ytt
  CALL @print_msg
  LDELM @str_talk_rat
  CALL @print_msg
  RET
```

```
talk_snake:
  LDELM @str_ytt
  CALL @print_msg
  LDELM @str_talk_snake
  CALL @print_msg
  RET
```

```
talk_spider:
  LDELM @str_ytt
  CALL @print_msg
  LDELM @str_talk_spider
  CALL @print_msg
  RET
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; talk strings
```

```
str_talk_where:
  .bytes " > Talk where? ", 13, 10, 0
```

```
str_TALK:
  .bytes " TALK ", 0
```

```
str_ytt:
    .bytes " You talk to ", 0

str_talk_air:
    .bytes "the ", 13, 10, " air. But no-one", 13, 10, " is there!", 13, 10,
0

str_talk_wall:
    .bytes "the ", 13, 10, " wall. It's not ", 13, 10, " listening.", 13,
10, 0

str_talk_cdoor:
    .bytes "the ", 13, 10, " closed door. ", 13, 10, 32, 34,"Open sesame!",
34, 13, 10, 0

str_talk_odoor:
    .bytes "the ", 13, 10, " open door. Nothing happens.... YET!", 13, 10, 0

str_talk_rat:
    .bytes "the ", 13, 10, " rat. It squeaks", 13, 10, " in anger!", 13, 10,
0

str_talk_snake:
    .bytes "the ", 13, 10, " snake. It hisses", 13, 10, " dangerously!", 13,
10, 0

str_talk_spider:
    .bytes "the ", 13, 10, " spider. It moves", 13, 10, " closer!", 13, 10,
0
```

From:

<https://www.appledog.ca/wiki/> - Appledog

Permanent link:

https://www.appledog.ca/wiki/doku.php?id=sd:rogueima_i_mvp-4&rev=1778343098

Last update: **2026/05/09 16:11**

