# Stardodger III – the assembler version

## Stewart Russell shows you how to program the same game three times in three very different languages

**BCPL produced a game that would be fast enough for anyone, but was about 10k long. This seemed quite excessive, so a non-singing, non-dancing compact game in Z80 code was developed in Assembly language.**

Assembly language programming is quite a different kettle of fish from Basic or BCPL, with absolutely no safety net provided for the unwary. Simple housekeeping tasks such as printing a string have to be written by the author and these can frequently prove to be the most difficult bits.

Again, the Basic Stardodger served as the template for this model, with slight differences. The screen numbers in this version are printed in hexadecimal simply because a hex routine is shorter to write than a decimal one.

Reasonable care has been taken that each routine does not corrupt registers unnecessarily. This wastes space in the form of PUSHes and POPs, but the approach is preferable to a possible system crash.

The random routine and the screen drawing routine could both be made shorter, but they work, this being of the utmost importance. Results of tests were not generally returned in registers but in memory locations, because it is all too easy to corrupt a register by mistake.

The listing was produced using the Maxam rom, so the long label names may cause problems for ADAM, Devpac, Zapp and Code Machine users, but are there to increase readability. Check your manuals to see how label names should be entered into your particular assembler.

Your reward for all of this effort? A paltry 937 bytes of code. This can be saved by hopping into Basic and typing:

```
SAVE"STARDOJ",B,&5000,&3A9,&5000
```

Some people may find this Assembler version a bit fast. Remedy this by changing part of the START subroutine from *LD A,3:CALL HAUDON* to *LD A,value:CALL HAUDON* where *value* can take any value from zero to 255 – 3 (default), 4, 5 or 6 (very slow) are the best ones to use. HAUDON, incidentally, is Glaswegian for "hold on".

### Conclusions

If there has to be a conclusive winner in the general niceness stakes, BCPL would take it by a short head. It was as simple as the Basic version to write, yet ran as quickly as the Assembly version. Its only drawback was the length of the object file produced.

Producing the Assembly version was certainly challenging, but the effort involved doesn't really justify the result.

Although coming last in the race, Basic provided the slow, good-natured workhorse from which the tetchy thoroughbreds were descended.

My current record on both the BCPL and the Assembly versions is Screen 11. I've made it through Screen 19 on the Basic one. Can you get further? Go forth, and dodge them stars!

| | Basic | BCPL | Assembly |
|---|---|---|---|
| Routine | lines | procs | procs |
| Initialisation | 20-70 | start | su |
| Print title screen | 90-170 | start | tiscr |
| Draw game screen | 180-450 | drawscr | drawscr1, plot |
| Main game logic | 470-530 | start | start |
| Print game over screen | 550-600 | start | losscr |
| Print success screen | 620-680 | start | wonscr |
| Wait for keypress routine | 700-760 | waitkey | waitmess, waitkey |

*A comparison of the main routines*

```
;Stardodger by Stewart C Russell

;Standard Amsdos equates
gra_initialise equ #bbba
gra_move_abs   equ #bbc0
gra_move_rel   equ #bbc3
gra_line_rel   equ #bbf9
gra_set_pen    equ #bbde
gra_test_rel   equ #bbf3
gra_wr_char    equ #bbfc
scr_initialise equ #bbff
scr_set_ink    equ #bc32
scr_set_border equ #bc38
scr_set_mode   equ #bc0e
km_initialise  equ #bb00
km_read_key    equ #bb1b
km_test_key    equ #bb1e
txt_initialise equ #bb4e
txt_output     equ #bb5a
txt_set_cursor equ #bb75
kl_time_set    equ #bd10
kl_time_please equ #bd0d

org #5000

.start    call su            ;the game itself
          call su            ;set up colours etc.
.oloop    call tiscr         ;print title screen
.loop1    call drawscr1      ;draw screen box
          call plot          ;plot stars
          ld de,2
          ld hl,200
          call gra_move_abs  ;move to start position
.contscr  call shift         ;test if shift is pressed
          ld a,3
          call haudon        ;0.01 second delay
          ld a,(dy)          ;get result of shift test
          cp 4               ;should it go up?
          jr nz,down         ;draw it down if it isn't
          ld hl,4
          ld de,4
          call gra_line_rel  ;draw line upwards
.doline   call test          ;have we hit anything?
          ld a,(col)         ;get result of collision test
          cp 1               ;is it white?
          jr z,lost          ;if it is, end game.
          cp 3               ;have we hit the gap?
          jr z,next          ;if so, go to next screen.
          jr contscr         ;continue the screen if neither.
.lost     call losscr        ;print 'you goofed' etc.
          ld a,1
          ld (scr),a         ;first screen number = 1
          ld hl,5
          ld (q),hl          ;first no of stars = 5
          ld a,4
          ld (dy),a          ;initial direction = up
          jr oloop           ;go back to start of outer loop
.next     call wonscr        ;print 'well done' etc.
          jr loop1           ;go to the next screen
.down     ld hl,-4
          ld de,4
          call gra_line_rel  ;draw line downwards
          jr doline          ;return to main loop

;draws the lines for the playing screen
.drawscr1 ld a,1
          call scr_set_mode  ;mode 1
          ld a,1
          call gra_set_pen   ;graphics pen 1
          ld de,629
          ld hl,0
          call gra_line_rel  ;drawr 629,0
          ld de,0
          ld hl,170
          call gra_line_rel  ;drawr 0,170
          ld de,0
          ld hl,60
          call gra_move_rel  ;mover 0,60
          ld de,0
          ld hl,169
          call gra_line_rel  ;drawr 0,169
          ld de,-629
          ld hl,0
          call gra_line_rel  ;drawr -629,0
          ld de,0
          ld hl,-399
          call gra_line_rel  ;drawr 0,-399
          ld de,0
          ld hl,2
          call gra_move_rel  ;mover 0,2
          ld de,627
          ld hl,0
          call gra_line_rel  ;drawr 627,0
          ld de,0
          ld hl,168
          call gra_line_rel  ;drawr 0,168
          ld de,0
          ld hl,60
          call gra_move_rel  ;mover 0,60
          ld de,0
          ld hl,167
          call gra_line_rel  ;drawr 0,167
          ld de,-625
          ld hl,0
          call gra_line_rel  ;drawr -625,0
          ld de,0
          ld hl,-399
          call gra_line_rel  ;drawr 0,-399
          ld a,3
          call gra_set_pen   ;graphics pen 3
          ld de,637
          ld hl,0
          call gra_move_abs  ;move 637,0
          ld de,0
          ld hl,400
          call gra_line_rel  ;drawr 0,400
          ld de,2
          ld hl,0
          call gra_move_rel  ;mover 2,0
          ld de,0
          ld hl,-400
          call gra_line_rel  ;draw second line at screen
          ld a,1             ;edge for completion test.
          call gra_set_pen   ;graphics pen 1
          ret

;returns A=0 and NC if no key - key no. and C if ok.
```

```
.keyvalid or a              ;clear carry flag
          call km_read_key
          ret c             ;return if we have a key
          xor a             ;clear accumulator to 0
          ret


;prints string addressed by DE at text position H,L.
;string must end with zero byte.
.prts     push af
          push hl
          push de           ;save registers
          call txt_set_cursor;set cursor to H,L
          pop de            ;get string address
.pl       ld a,(de)         ;get character
          inc de            ;increment string pointer
          or a              ;is character zero?
          jr z,pret         ;go to pret if it is
          call txt_output   ;print it
          jr pl             ;loop to start of routine
.pret     pop hl
          pop af            ;restore registers
          ret


;pauses until a key is pressed
.waitkey  push af
.w        call keyvalid     ;loop while key pressed
          jr c,w            ;to clear buffer.
.u        call keyvalid
          jr nc,u           ;loop until key pressed.
          pop af
          ret

;pauses for A/300ths of a second.
.haudon   push de
          push hl
          push af
          ld de,0
          ld hl,0
          call kl_time_set  ;reset clock
          pop af            ;get pause length
.tl       call kl_time_please;get time
          cp l              ;is it equal to value in A?
          jr nz,tl          ;loop while it isn't
          pop hl
          pop de
          ret

;get shift state and return 4 or -4 in (dy)
.shift    push af
          push hl
          push bc
          ld a,21
          call km_test_key  ;has shift been pressed?
          jr nz,yep         ;if so, go to yep
          ld a,-4
.nup      ld (dy),a         ;store value of key test
          pop bc            ;for line increment.
          pop hl
          pop af
          ret
.yep      ld a,4            ;positive line increment
          jr nup

;prints 'Press any key to continue' at 8,25
.waitmess ld hl,#0819
          ld de,paktc
          call prts
          ret

;print the title screen
.tiscr    ld a,1
          call scr_set_mode ;mode 1
          ld hl,#1001
          ld de,sdr
          call prts         ;print title
          ld hl,#105
          ld de,avoi
          call prts         ;print instructions 1
          ld hl,#906
          ld de,wond
          call prts         ;print instructions 2
          ld hl,#c0d
          ld de,uses
          call prts         ;print key to use
          call waitmess
          call waitkey      ;wait for a keypress
          ret

;prints A in hex.
.prta     push af
          push bc
          ld b,2            ;loop counter for two nibbles
          ld c,a            ;store A
          rra
          rra
          rra
          rra
.gnyb     and #f            ;get nibble
          cp #a             ;is it >=10?
          jr nc,hex         ;if so, go to hex
          add a,#30         ;convert to ascii char 0-9
.prn      call txt_output   ;print it
          ld a,c
          djnz gnyb         ;act on lower nibble
          pop bc
          pop af
          ret
.hex      add a,#37         ;convert to ascii char a-f
          jr prn

;tests point relative (2,dy/2) and returns in (col).
.test     push bc
          push de
          push hl
          ld a,(dy)         ;get players line direction
          cp 4              ;is the line currently going up?
          jr nz,n4          ;go to n4 if it isn't
          ld hl,2           ;use relative coords (2,2) for
          ld de,2           ;test if going up.
.ctest    push hl           ;save HL
          call gra_test_rel ;test point relative to line end
          ld (col),a        ;store ink result
          pop hl            ;restore HL
          call nhl          ;negate HL
          ld de,-2          ;move graphics cursor back
          call gra_move_rel ;to original position.
          pop hl
          pop de
          pop bc
          ret
.n4       ld hl,-2
          ld de,2           ;use relative coords (2,-2)
          jr ctest          ;for test if going down

;negate HL
.nhl      push af
          ld a,h
          cpl
          ld h,a            ;complement H
          ld a,l
          neg
          ld l,a            ;negate L
          pop af
          ret

;prints success screen
.wonscr   ld a,1
          call scr_set_mode ;mode 1
          ld hl,#1001
          ld de,well
          call prts         ;print 'Well Done'
```

```
        ld hl,#90d
        ld de,stan
        call prts           ;print next screen message
        ld a,(scr)          ;get next screen number
        call prta           ;print screen number in hex
        call waitmess
        call waitkey        ;press any key to continue
        ret


;prints lose screen
.losscr ld a,1
        call scr_set_mode   ;mode 1
        ld hl,#1001
        ld de,youg
        call prts           ;print 'You Goofed'
        ld hl,#40d
        ld de,numb
        call prts           ;print number completed message
        ld a,(scr)          ;get no of next screen
        dec a
        dec a               ;reduce it to screens completed
        call prta           ;print no of screens completed
        call waitmess
        call waitkey
        ret


;returns a psuedo-random integer in HL
.random push af
        push bc
        ld hl,(seed)        ;get seed
        push hl             ;save it
        add hl,hl
        add hl,hl
        add hl,hl
        add hl,hl
        add hl,hl
        add hl,hl
        add hl,hl
        add hl,hl
        add hl,hl           ;HL=seed multiplied by 256
        pop bc              ;get original seed
        add hl,bc           ;add seed to (seed*256)
        ld bc,41
        add hl,bc           ;add 41 to result for fun of it
        ld a,r              ;get value of memory refresh
        add l               ;add low byte of random result
        ld l,a              ;return this value to HL
        pop bc              ;restore BC
        adc hl,bc           ;add BC to HL (fiddle factor)
        ld (seed),hl        ;store result for later use
        pop af              ;restore AF
        ret


;remainder routine, returns HL = HL mod DE.
.rem    or a                ;clear carry flag
        sbc hl,de           ;is DE>HL?
        add hl,de           ;restore HL
        ret c               ;return if DE>HL
        sbc hl,de           ;HL=HL-DE(-0 in carry flag)
        jr rem              ;recurse to start

;plots the relevant number of asterisks.
;keeps track of screen no and no of stars
;in scr (byte) and q (word) respectively.
.plot   push af
        push bc
        push de
        push hl
        ld hl,(q)           ;get number of asterisks
.pltlp  push hl             ;save loop counter
        call random         ;get random number in HL
        ld de,561
        call rem            ;HL = HL mod 561
        ld de,50            ;add 50 to it to allow for
        add hl,de           ;initial reaction time.
        push hl             ;save x-coord
        call random         ;get random number for y-coord
        ld de,361
        call rem            ;HL = HL mod 361
        ld de,20
        add hl,de           ;add 20 for tidyness
        pop de
        call gra_move_abs   ;move to the random position
        ld a,'*'
        call gra_wr_char    ;and write an '*' there
        pop hl              ;get loop counter
        dec hl              ;decrement it
        ld a,h
        or l                ;is it zero?
        jr nz,pltlp         ;loop if not
        ld a,(scr)          ;get screen no.
        inc a               ;increment screen number
        ld (scr),a          ;restore new screen number
        ld hl,(q)           ;get number of asterisks
        inc hl
        inc hl
        inc hl
        inc hl
        inc hl
        ld (q),hl           ;set no of stars (scr*5)
        pop hl
        pop de
        pop bc
        pop af
        ret

;set up mode, colours, etc
.su     ld a,1
        call scr_set_mode   ;mode 1
        call txt_initialise
        call km_initialise
        call gra_initialise
        call scr_initialise
        xor a
        ld b,a
        ld c,a
        call scr_set_ink    ;ink 0,0
        ld a,1
        ld b,26
        ld c,b
        call scr_set_ink    ;ink 1,26
        ld a,3
        ld b,0
        ld c,b
        call scr_set_ink    ;ink 3,0
        xor a
        ld b,a
        ld c,a
        call scr_set_border;border 0
        ret

;static variables and strings

.dy     defb 4
.col    defb 0
.scr    defb 1
.paktc  defm "Press any key to continue.",0
.sdr    defm "Stardodger",0
.avoi   defm "Avoid the killer asterisks, and seek the",0
.wond   defm "wondrous Nextscreen Gap.",0
.uses   defm "Use SHIFT to climb",0
.well   defm "WELL  DONE",0
.stan   defm "Stand by for screen ",0
.youg   defm "YOU GOOFED",0
.numb   defm "Number of screens completed = ",0
.seed   defw #abcd
.q      defw 5

end
```