Wildbits classroom recap

Presented by Michael Weitman @ VCF EAST April 17-19, 2026

Wildbits Computing Company descended on VCF East this April with **fifteen Jr2 machines** and a curriculum primarily focused on the default 6502 & SuperBASIC configuration. The class leveraged the current GA 1x core and a fresh (April 14th) release of SuperBASIC.

Instruction took the form of 4 sessions per day across 2 full days. The goal was to create a starting point, empowering participants with a platform for exploration (or just play games or to create) including graphics editors and SuperBASIC code examples. But some came specifically for the lecture and discussion. Sunday was dedicated to the 6809-based NitrOS-9 operating system and BASIC09.

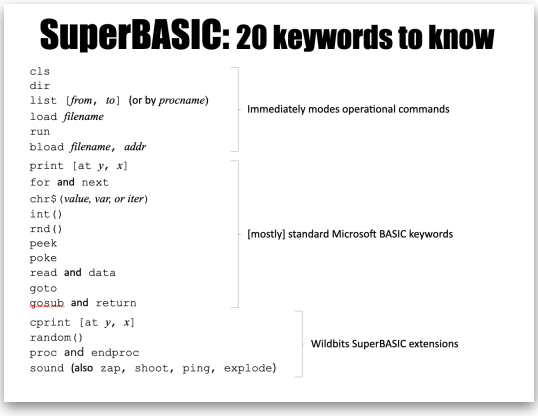
Session 1: Intro to Wildbits K2/Jr2 & SuperBASIC with character graphics

The Wildbits K2/Jr2 has plenty to offer and is a good fit for VCF East's diverse crowd, so it was appropriate to start at *the beginning,* introducing the 6502 CPU and its origin story. Tie-ins to vintage systems sharing architectural attributes and references to famous people and events in history were woven into discussion.

After a port-by-port and hardware feature overview, Dartmouth's first version of the BASIC programming language was briefly discussed. From there, we quickly moved to SuperBASIC, beginning with general navigation and features.

With the introduction complete, machines were powered-up and we examined the alpha-numeric and graphic character sets with a few exercises, including printing with formatting, with colored characters and backgrounds, and discussing the famous Commodore **10 PRINT** example, before moving onto the *Card** program.

Several slides were presented including the "20 Keywords to know" slide. A supporting discussion on the basics of line numbers, how to list programs, anatomy of the `for` / `next` loop, and tangental material was discussed. As a lead-in to graphics, the *invaded* demo was put into action (it uses redefined character graphics and the `sound` command - see pg. 9 for this listing).



* "card.bas" is a 24-line SuperBASIC example that demonstrates BASIC language programming concepts such as variable assignment, looping, printing with graphic characters and control characters; it also uses SuperBASIC procedures (`proc` / `endproc`) . See pg. 9 for a program listing.

A deep dive into Cards is dealt in the "*Unofficial Wildbits Getting Started Guide - Volume 1"* on pages 21-26. Volume 1 is a 101-page spiral bound book intended to walk beginner and intermediate users through dozens of exercises and Wildbits features. Look for it in June 2026.

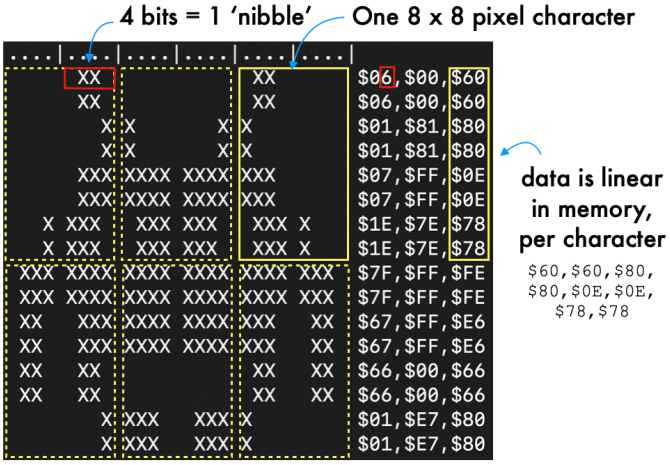
This left time for students to experiment and play with binaries including some famous game titles and the Wildbits Graphics Toolkit (written by econtrerasd).

From session to session, the curriculum was tweaked based on student interests and the number of students attending each class. Of the 15 stations, the most busy class saw 10 or 11 students but volunteers were more than happy to jump in. Smaller sessions presented opportunities for active labs and one-on-one time to ensure that everybody got a good taste of what the Jr2 platform is capable of.

Session 2: SuperBASIC Sprite & Bitmap graphics including 6502 Assembly

The 2nd session of each day began with a brief look at computer graphics from an Apple II bitmap screen perspective (280 x 192 resolution). This framed a premise; 1 MHz systems struggled mightily to manage graphic data with such a slow CPU!

Shape tables were mentioned as a corollary to redefined graphics introduced in the prior session and the Commodore port of the game Omega Race for the VIC-20's 22 column by 23 row screen was cited as a gold-standard example of the use of pre-calculated shapes. This led to a discussion of the Commodore 64's gift to gaming (sprites), also known as hardware accelerated movable objects. For those unaware of the specifics, the Commodore boasted 8 * 24 x 21 pixel movable objects.



Using the *Balloon Gone Wild Setup* program, students were instructed to initiate sprite-load, and then manually move (using `sprite n image i to x, y`) a balloon-shaped graphic object to mid-screen.

After zero'ing (blacking out) the color `palette`, colors were brought back, one-by-one. A discussion of 24-bit color, Pantone, and a method to cycle colors was discussed before `goto` and `proc`-based versions of the program were executed.

Other sprite related examples included the *Halloween Jack* mini-game (pg. 11-17), Celton's *Balls* program (which multiplexes 128 sprites on-screen with a complex collision algorithm), and experimentation with the *Wildbits Sprite Editor*. As a bridge to bitmaps, the *Simulation Complete*[*] program was exercised and discussed.

* "simulation complete.bas" is a 51-line SuperBASIC example that uses 24 lines of embedded pixel sprite `data` to demonstrate randomized bitmap drawing through an array of (up to) 64 alien 'agents'. The result is a map-like mosaic, rendered in checkerboard and solid patterns. See code on page 10.

Students were encouraged to modify the `num_aliens` constant, examining the effect, and to modify alien images, color, or program logic. This programming example uses three, 64 element arrays to track and manage graphic object attributes.

Finally, two multi-layer graphic demo applications were introduced:

- Mu0N's *gfxDemo*; probably the best known example of easy to control graphic layer priority (including tiles). gfxDemo is also a publicly available reference example of the C Programming Language source for the Wildbits platform.
- econtrerasd's *rpg-demo.bas* features a joystick controllable main player character (Valkyrie) on a quest for knowledge (literal signposts, highlighting Wildbits features). This program uses layers and sprite animation.

And playtime would be complete with access to top-quality games; in this case, Pitfall, Lode Runner, H.E.R.O, Trick-or-Treat, Race, and more. Browsing and navigation was made easier with Micah's *FM* (File Manager). Each machine had an SD card loaded with binary executables and SuperBASIC examples.

With so much action and content, none of the early classes had time for the planned inline assembly language portion of the curriculum, but there was a 30 minute discussion focused on 6502 assembly language, and specifically, a showing of Stephen Edwards *Advanced 6502 Assembly Programming for the Apple II* during the advanced development session (see below).

Session 3: Intro to Sound on the Wildbits K2/Jr2 system "from SID to MIDI"

The sound and music session was the most varied, not because of the end-to-end range of K2/Jr2 capabilities, but because of the diversity of attendee backgrounds and interests.

Each year, VCF East selects a show theme and this year it was "Dawn of the PC"; this drew PC gamers with their love of PC sound from ISA cards to the Roland MT-32. If you know PC history, you'll recall that 1st gen machines had the same 'sound' as the 8-bit systems that came 5 years prior; beeps from a tiny speaker. The PC's ISA based bus, however, invited expansion, and it wasn't long before companies such as Creative Labs, Adlib, and Gravis, created products such as incarnations of the SoundBlaster, Gravis Ultrasound, and the Wave Blaster add-on.

First generation sound cards used commonly available synth chips (such as Yamaha's OPL2, OPL3, and others), however, the end-user rarely invested time to gain access to them from a programming perspective. Instead, they were twice removed from the hardware, configuring the card from a physical (jumper) perspective, and then adding a device driver and/or telling the game they wanted to play to use it. Users were less concerned about the details of how it all worked at the chip or driver level, they just heard that card 'a' sounded better than card 'b' and had to buy one.

On the Wildbits Jr2 machine, the opposite is true. Programmers have direct, register and byte-level access to the hardware and a wide array of technology to choose from including general MIDI and wavetable synthesis onboard.

All of this background is to share that after an initial discussion with the class, we talked about the origins of discreet timer based circuits (found in famous arcade games of the late 70s). To illustrate the point, we passed around a small circuit on a perf-board, cloned from the Space Invaders schematic (which used the Ti 76477 IC, responsible for the famous UFO sound).

The natural progression from this origin was the Ti 76489 4-voice integrated circuit, commonly known as the PSG; a great friend of SuperBASIC.

SuperBASIC provides native support through the `sound` command and our first exercise was to examine the sound effects from the Halloween Jack mini-game which uses the PSG for a tonal melody and some arcade sound effects as follows:

Jumping sound: `sound 1,213,16,-4`
Damage sound: `sound 3,7*16,8`
Game over sound: `sound 0,142,96,2`

voice # [0..3]
pitch *value**
duration (ticks)
freq bend rate

After passing out ear pods (we brought 40 pairs), we tortured the students with some SID music before bringing up Mu0N's FireJam for experimentation.

We also shared dwsJason and digarok's Xmas_24 demo, featuring synthesized "Ho Ho Ho" speech and MOD music (a crowd pleaser on many levels).

Additional content on SuperBASIC's `sound` and `playing` keywords were presented through excerpts and diagrams from "Getting Started - Volume 1".

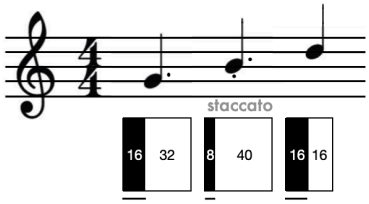
The math required to convert from actual frequency to pitch-value was discussed (see below*), and a handful of note sequences were examined including the famous Pitfall *rope-over-lake* sound:

```
sound 2,853,48
sound 2,509,48
sound 2,426,16
sound 2,509,48
```

Halloween Jack's melody was further examined (frequencies and timing mapped to values held in `data` statements), and the `playing` queue feature was discussed.

Finally, we shifted gears to the dream.fr SAM2695 IC and introduced General MIDI, use of instruments for game sound effects, and converted a SuperBASIC "play all instruments" program to work with the SAM IC via `poke $dda1, {byte}`

The following graphic illustrates this cycle using the timing values from the first three notes of the song:



Timing data is embedded in data statements for each 'event'; an event may be a note playing at a given frequency for a length of time (in ticks) or no sound for a given length of time.

To add dynamics to this simple melody, some of the notes are played with a ratio of 1/3rd note-on, and 2/3rds, note-off; some notes (such as the 3rd note above) have equal time (16 tics and 16 ticks), and between the two, a 1/6th and 5/6ths time split (staccato).

* Pitch-*value* is not the same as frequency, but we can calculate it easily with: 111,563/frequency. As an example, 111,563/**440 (Hz)** otherwise known as a 4th octave 'A', equals 253.5. We will need to round the final number since `sound` only accepts integers; therefore, we end up with a value of 254.

Session 4: Advanced development on the Wildbits K2/Jr2 system

The last session of each of the full days (Friday and Saturday 3:30pm-5:20pm) focused on software development and any combination of 6502/65816 Assembly Language, off-host SuperBASIC development (aided by `xgo`/`xload`), and/or C Programming, were all topics up for discussion.

People wandered in looking for everything from "BASIC without line numbers" to IBM XT 286 Assembly Language development. This last request seemed peculiar, at first.

It came about after surveying the 8 attendees to find, not just one pair of people interested in 286 development, but two! (a father/son team). The father took some 8086 assembly language in college (in the 80s, assumed), and the son had just taken a RISC V class. Still another person was a professional TypeScript/REACT developer with no microcomputer or Assembly language experience.

We were able to make the best of it, walking the class through a white-board discussion of a simple `.asm` example by first demonstrating how to address screen memory in SuperBASIC to print the letter 'A' in the upper left of the screen:

```
poke 1,2
poke $c000, $41
```

Instruction-set aside, any discussion of Assembly Language must begin with the definition of *registers*, outlining the basics of data bus width, and by association, the scale of values that can be managed without heroics. The impromptu session began with examples of different classes of opcodes (the classic five):

load	`lda` or `ldx {immediate_mode_value}`
store	`sta {zero_pg_addr}` or `{absolute_addr}`
increment	`inx` (increment x)
store indexed	`sta {screen_addr}, x`
branch on condition	`bne {relative_addr}`

After pulling up the projector screen to reveal the white board, we hand-wrote and refined what became the following code:

```
                *=2000
start           lda #$02
                sta $01
                ldx #$00
                lda #$41
loop_point      sta $c000,x
                inx
                bne loop_point
                brk
```

One participant was interested in further learning and we shared book references for the 6502/65816 including: Lance A. Leventhal's: 6502 Assembly Language Programming; Leventhal's: 6502 Assembly Language Subroutines, and David Eyes' Programming the 65816 including the 6502, 65C02, and 65802. (Leventhal's books are available from archive.org; Eyes' is on Amazon)

We closed the session with a discussion on Apple II graphics and an Assembly Language demo program developed by Stephen Edwards. Tethered to a cell phone, we were able to watch excerpts of Stephen's YouTube video on the subject.

Though it wasn't quite ready by Sunday, I discussed the rewriting of the main loop of the *Simulation Complete* demo in Assembly language (we will cover this in a future newsletter).

Other advanced topics and noteworthy content

NitrOS-9 and BASIC09: On Sunday, Matt Massie and Boisy Pitre spent two hours walking participants through BASIC09 and NitrOS-9 updates.

Matt began by demonstrating several BASIC09 features including editing and loading a program, 'packing' a program for execution, bouncing back and forth from a shell, use of the 'WILD' module for drawing bitmap graphics, and some background on the memory allocation methodology.

He also talked about his WizFi project including a debugging feature which formatted serial data in a useful debug style monitor-like format.

Other live demonstrations included Living Worlds, a Paint application, JFed's text editor, and R Taylor's `play` command (a multi-format music player application).

Finally, Matt gave a demo of his Sprite Editor using a set of Centipede sprites that he had been working on for use in his BASIC09 proof-of-concept game.

Boisy anchored the 2nd hour, discussing the history of OS-9 and NitrOS-9. His years of experience always brings a few nuggets of knowledge to the masses:

- an anecdote on compatibility of NitrOS-9 to support the Hitachi 6309 CPU
- the cost of OS-9 level 2 (if purchased off the shelf at Radio Shack back in the 80s) was a mere $79 and that included BASIC09; the original cost of level 1 was $99, without BASIC09 (which was $99 additional)
- Microware charged several hundred dollars, if not more, for the same software (when sold to a commercial entity)

During Boisy's portion of the discussion, the differences between release levels and the design philosophy to use both on the Wildbits platform were highlighted along with an explanation of the FEU (Foenix Executive Utility) within.

Boisy also discussed the genesis and capabilities of his DriveWire project, ports of NitrOS-9 to other platforms he has worked on across the years, and the circumstances behind his becoming aware of the Foenix project in the first place.

For the uninitiated, running NitrOS-9 on a Jr2 will take over your system. It requires an alternate FPGA core in addition to consuming flash blocks and a linux/unix `dd` formatted SD image. On the K2 however, NitrOS-9 can happily reside in one of the four context slots, next to the standard 6502 and SuperBASIC context; and this is dip-switch selectable at boot-time.

NitrOS-9 is attractive because it is a *real* operating system with binary memory re-use, an interactive shell, dozens of system and user-level programs, the ability to leverage an on-platform C complier, Assembler and monitor, and BASIC09. Level 2 adds task management, preemptive multi-tasking, and more!

There is ongoing work to enhance the Wildbits port with a multi-instance terminal capability (ability to switch back and forth between different text screens), extensions to the WILD module, and new user programs. And this is all made possible through a core group of a half-dozen dedicated developers.

These two sessions were the only two that we were able to record on video; they will be released along with the redux series of videos in the June 2026 timeframe.

Recent SuperBASIC enhancements: The classroom Jr2 machines had the benefit of a brand new version of SuperBASIC (v1.2) and with it, plenty of quality-of-life improvements to share with attendees.

Since many of those attending had experience with vintage ATARI, Commodore, CoCo, Sinclair or other platforms, many of the new features felt futuristic, even for 2026!

File under the category of "did you know":

- pressing/holding shift will temporarily pause program or directory listings
- pressing the PC keyboard INSERT key will 'open up' lines of whitespace; useful to insert lines of code or execute immediate mode expressions
- pressing the WLD / '*F*' / START key plus cursor UP/DOWN scrolls SuperBASIC code upwards or downwards, bringing new lines from above or below the screen
- it is now possible to have a SuperBASIC program that is well larger than the 'leftover' space within the first 64K of memory. With SuperBASIC v1.2, memory is allocated beginning at $06:0000 using numerous 8K blocks (managed by the MMU); source can be 128K or more, with arrays, sprite and tile data, and the bitmapped screen data managed elsewhere!

There are also stability enhancements and new keywords and directives such as:

usage	example
`print at y, x`	`print at 15, 10; "#"`
`input at y, x`	`input at 0, 0; "Guess a number (1-10): "; num`
`screen(y, x)`	`print screen$(15, 10)`
`screen$(y, x)`	`ch = screen(15, 10)`

Four great YouTube videos: One, a promotional game-jam reel, two Core2x feature demos, and one instructional video are worth highlighting. These artifacts were shown across different sessions of the Wildbits VCF Classroom.

"October 2024 Game Jam - Hype video" - contains a mash-up of game submissions, most adopting the spooky Halloween theme. It is a good demo of video and gameplay capabilities, demonstrating sprites, tiles, bitmaps, and redefined characters. We played this several times.

Running time: 2:12

John Baker's "VS1053b Geometry Kernel demo" - leverages a DSP plug-in within the systems VLSI IC to render vector objects in a 3d space, passing back a list of vertices to graph with the Core2x FPGA based line-draw. This demo is especially impressive graphically and gifts the Jr2 & K2 with a vector-engine-like capability as used on ATARI arcade machines (Asteroids, Battle-Zone, Tempest).

Running time: 3:17

John Baker's "Memory Text Animation demo" - leverages the new (Core2x) Memory Text (aka memtext) layer, streaming rendered high-color palette video from SD storage. This example uses a 512 color palette across 1024 glyphs resulting in high-color 640x480 pixel video replay. Amiga, eat your heart out!

Running time: 6:37

"Stephen Edwards - Advanced 6502 Assembly Programming for the Apple II" - a concise master class of the 6502 theory of operation, instruction set, introduction to Apple II graphic and shape table examples and a bouncing-ball program. This was shared during the advanced development afternoon sessions. Stephen's program was ported to the F256 machine and used to support a discussion on memory management and graphics. A .pdf document can be downloaded here.

Running time: 33:57

There was plenty learned about how to make a future event better, beginning with spending time to canvas and pre-book attendees. VCF East is a quick 2.5 day event and people scramble around campus; not the most conducive environment for a classroom based on a machine that few heard about. But it all starts here.

I would like to thank A.J. Griglak for all of his efforts; he was in the room, not only for a full day of setup, but for 95% of the remainder of the live sessions and stayed on site well beyond my departure on Sunday evening. I also want to thank Dean Notarnicola for his participation, and Mike Cassera for bearing the expense and trouble to design, fab, build, and ship the joystick 'hats' to me in time for the class. Also to Matt and Boisy; I learn something new about NitrOS-9 every time we meet. Thanks also to Jeffrey Brace (show runner) and his team, and Aleksey and his family for entrusting me to represent his products. More soon…

card.bas

```
10    ul=188:ur=189:ll=190:lr=191:cs=252
20    up=16:rt=6:dn=14:vb=130:hb=150
30    width=7:height=9:cls
40    cprint chr$(ul);:draw_chrs(width,hb)
50    cprint chr$(ur)
60    for y=1 to height-2
70      cprint chr$(vb);
80      draw_chrs(width,32)
90      cprint chr$(vb)
100   next
110   cprint chr$(ll);
120   draw_chrs(width,hb):cprint chr$(lr)
130   for x=1 to height-1
140     print chr$(up);
150   next
160   print chr$(rt);"A";:cprint chr$(cs);
170   print chr$(dn);chr$(dn);chr$(dn);
180   cprint chr$(cs);:print chr$(dn);chr$(dn);
190   print chr$(dn);:cprint chr$(cs);"A":end
500   proc draw_chrs(n,char)
510     for x=1 to (n-2)
520       cprint chr$(char);
530     next
540   endproc
```

invaded.bas

```
100   print chr$(131);chr$(144):cls :freq=800
110   load_chardata()
120   poke 1,2:poke $C000,0:poke $C001,1:poke $C002,2
130   poke $C050,3:poke $C051,4:poke $C052,5
140   for x=0 to 70
150   step_alien()
160   next :explode :explode :explode :end
500   proc load_chardata()
510   poke 1,1
520   for x=0 to 47
530   read byte
540   poke $C000+x,byte
550   next
560   poke 1,0
570   endproc
600   proc step_alien()
610   poke $C000+3+x,peek($C000+2+x)
620   poke $C050+3+x,peek($C050+2+x)
630   poke $C000+2+x,peek($C000+1+x)
640   poke $C050+2+x,peek($C050+1+x)
650   poke $C000+1+x,peek($C000+x)
660   poke $C050+1+x,peek($C050+x)
670   poke $C000+x,32
680   poke $C050+x,32
690   for y=1 to (2500-(x*30)):next
700   sound 0,freq,4
710   freq=freq+60:if freq=1040 then freq=800
720   endproc
1000  data $06,$06,$01,$01,$07,$07,$1E,$1E
1010  data $00,$00,$81,$81,$FF,$FF,$7E,$7E
1020  data $60,$60,$80,$80,$E0,$E0,$78,$78
1030  data $7F,$7F,$67,$67,$66,$66,$01,$01
1040  data $FF,$FF,$FF,$FF,$00,$00,$E7,$E7
1050  data $FE,$FE,$E6,$E6,$66,$66,$80,$80
```

simulation complete.bas

```
10    cls :sprites on :bitmap on :bitmap clear 0:num_aliens=64
20    col=24:xfer_addr=$7800
30    for y=0 to 7
40      read line_data$
50      for x=0 to 7
60        addr=xfer_addr+(y*8)+x+3
70        pixel$=mid$(line_data$,x+1,1)
80        if pixel$="."
90          poke addr,0
100       else
110         poke addr,col
120       endif
130     next
140   next
150   poke $7800,$11:poke $7801,0:poke $7802,0
180   col=59:xfer_addr=$7843
190   for y=0 to 15
200     read line_data$
210     for x=0 to 15
220       addr=xfer_addr+(y*16)+x+2
230       pixel$=mid$(line_data$,x+1,1)
240       if pixel$="."
250         poke addr,0
260       else
270         poke addr,col
280       endif
290     next
300   next
310   poke $7843,1:poke $7844,0:poke $7945,$80
320   memcopy $7800,326 to $30000
500   dim coord_x(64):dim coord_y(64):dim coord_i(64)
505   for spr=0 to (num_aliens-1)
510     coord_x(spr)=random(300)+8:coord_y(spr)=random(220)+8
520     coord_i(spr)=random(2)
530     sprite spr image coord_i(spr) to coord_x(spr),coord_y(spr)
540   next
605   for spr=0 to (num_aliens-1)
610     dire=random(2):if dire=0 then dire=-1
620     coord_x(spr)=coord_x(spr)+dire
622     if coord_x(spr)=0 then coord_x(spr)=1
624     if coord_x(spr)=307 then coord_x(spr)=306
630     dire=random(2):if dire=0 then dire=-1
640     coord_y(spr)=coord_y(spr)+dire
642     if coord_y(spr)=0 then coord_y(spr)=1
644     if coord_y(spr)=229 then coord_y(spr)=228
650     sprite spr image coord_i(spr) to coord_x(spr),coord_y(spr)
655     plot color 128 to coord_x(spr)+1,coord_y(spr)+1
660   next
670   goto 605

2000  data "...oo..."
2001  data "..oooo.."
2002  data ".oooooo."
2003  data "oo.oo.oo"
2004  data "oooooooo"
2005  data "..o..o.."
2006  data ".o.oo.o."
2007  data "o.o..o.o"

3000  data "................"
3001  data "................"
3002  data "................"
3003  data "................"
3004  data "..x.....x......."
3005  data "...x...x........"
3006  data "..xxxxxxx......."
3007  data ".xx.xxx.xx......"
3008  data "xxxxxxxxxxx....."
3009  data "x.xxxxxxx.x....."
3010  data "x.x.....x.x....."
3011  data "...xx.xx........"
3012  data "................"
3013  data "................"
3014  data "................"
3015  data "................"
```

halloween jack.bas (pg. 1 of 7)

```
100   rem "Halloween Jack - A SuperBASIC Programming Example v1.1"
110   rem "by Michael Weitman - an 8-bit Wall of Doom Production"
130   palette 1,255,255,255:palette 2,132,0,0:palette 3,118,0,0
140   palette 4,96,0,0:palette 5,0,0,144:num_sprites=9:sprite_dim=8
150   xfer_addr=$7800:poke xfer_addr,$11:xfer_addr=xfer_addr+1
160   for n=1 to num_sprites
170     poke xfer_addr,0:poke xfer_addr+1,0:xfer_addr=xfer_addr+2:loadsprite()
180   next
190   sprite_dim=24:poke xfer_addr,2:poke xfer_addr+1,0:xfer_addr=xfer_addr+2
200   loadsprite():poke xfer_addr,$80:sprites on :x_pos=160:y_pos=120
210   memcopy $7800,(num_sprites*66)+4+(24*24) to $30000
240   r_edge=315:l_edge=5:game_over=false :hurdle_speed=3:hurdle_x=320
250   wall_cntr=0:hit_pt=0:clean_pt=0:jmp_cntr=0:load_jump_ary():load_music()
260   bitmap on clear 15:cursor off :instruct()
280   sprite 0 image 3 to x_pos,y_pos
290   sprite 3 image 6 to x_pos+1,y_pos+1
300   sprite 4 image 8 to x_pos,113
800   while not(game_over)
810     if playing(0)=0
820       load_phrase()
830     endif
840     j_input=joyx(0)
850     if j_input
860       direc=j_input
870       if direc=1
880         if x_pos<r_edge
890           x_pos=x_pos+1
900           sprite 0 image 0 to x_pos,120
910           sprite 3 image 6 to x_pos-1,121
920           sprite 4 image 8 to x_pos,113
930         endif
940           else
950         if x_pos>l_edge
960           x_pos=x_pos-1
970           sprite 0 image 3 to x_pos,120
980           sprite 3 image 6 to x_pos+1,121
990           sprite 4 image 7 to x_pos,113
1000        endif
1010      endif
1020    endif
1030    if joyb(0)
1040      jump()
1050    endif
1060    move_hurdle(hurdle_speed)
1070    wait(1)
1080  wend
1090  show_score()
1100  cursor on
1110  end
```

```
1300  proc jump()
1310    rev=false
1311    side=0
1312    if x_pos>=hurdle_x
1314      side=1
1316    endif
1320    jmp_cntr=jmp_cntr+1
1330    sound 1,213,16,-4
1380    if direc=1
1390      jump_right()
1650    else
1660      jump_left()
1920    endif
1930    check_for_clean()
1988  endproc
2100  proc log_hit()
2110    hit_pt=hit_pt+1
2120    sound 3,7*16,10
2130    print chr$(16);"hit pts: ";hit_pt;" / clean jumps: ";clean_pt
2140  endproc
2300  proc move_hurdle(speed)
2310    counter=counter+1
2320    if counter>speed
2330      check_hurdle()
2340      counter=0
2350    endif
2360  endproc
2500  proc check_hurdle()
2510    if hurdle_x
2520      hurdle_x=hurdle_x-1
2530      sprite 1 image 4 to hurdle_x,122
2540      sprite 2 image 5 to hurdle_x,127
2550      if hit(0,2)
2560        log_hit()
2570      endif
2580    else
2590      sprite 1 off
2600      game_over=true
2610    endif
2620  endproc
2690  proc load_jump_ary()
2700    dim jumprx(15):dim jumpry(15):dim jumplx(15):dim jumply(15)
2710    for n=0 to 14
2720      read jumprx(n)
2730      read jumpry(n)
2740    next
2750    for n=0 to 14
2760      read jumplx(n)
2770      read jumply(n)
2780    next
2790  endproc
2800  proc clean_jump()
2810    sound 2,853,48
2820    sound 2,509,48
2830    sound 2,426,16
2840    sound 2,509,48
2850    clean_pt=clean_pt+1
2860    print chr$(16);"hit pts: ";hit_pt;" / clean jumps: ";clean_pt
2870  endproc
```

```
2890  proc load_music()
2900    dim note(54):dim time(54):dim ps(4):dim pe(4):ph_len=12:ph_num=-1
2910    ps(0)=0:pe(0)=ph_len-1:ps(1)=pe(0)+1
2920    pe(1)=ps(1)+ph_len:ps(2)=pe(1)+1
2930    pe(2)=ps(2)+ph_len:ps(3)=pe(2)+1
2940    pe(3)=ps(3)+ph_len:ps(4)=pe(3)+1
2950    pe(4)=53
2960    for x=0 to 53
2970      read note(x)
2980      read time(x)
2990    next
2995  endproc
3005  proc load_phrase()
3010    ph_num=ph_num+1
3020    if ph_num=5
3030      ph_num=0
3040    endif
3050    for x=ps(ph_num) to pe(ph_num)
3060      tm=time(x):tm=tm>>1
3070      sound 0,note(x),tm
3080    next
3090  endproc
3300  proc wait(ticks)
3310    st=timer()
3320    while (timer()-st<ticks)
3330    wend
3340  endproc
3360  proc loadsprite()
3370    for y=0 to (sprite_dim-1)
3380      read line_data$
3390      for x=0 to (sprite_dim-1)
3400        addr=xfer_addr+(y*sprite_dim)+x
3410        pixel$=mid$(line_data$,x+1,1)
3420        if asc(pixel$)>122
3430          poke addr,asc(pixel$)-121
3440        else
3450          if pixel$="_"
3460            poke addr,1
3470          else
3480            if pixel$="."
3490              poke addr,0
3500            else
3510              poke addr,asc(pixel$)
3520            endif
3530          endif
3540        endif
3550      next
3560    next
3570    xfer_addr=xfer_addr+(sprite_dim*sprite_dim)
3580  endproc
```

```
3700  proc show_score()
3710    cls
3720    wait(120)
3730    sound off
3740    sound 0,80,64,-1
3750    for y=y_pos-7 downto 0
3760      sprite 4 to x_pos,y
3770        for x=1 to 25:next
3780    next
3790    sprite 4 off
3800    wait(60)
3810    sound 0,142,96,2
3820    for y=y_pos to 240
3830      if y%4=0
3840        img=img+1&3
3850      endif
3860      sprite 0 image img to x_pos,y
3870      wait(1)
3880    next
3890    sprite 0 off
3900    sound 0,80,64,2
3910    for y=y_pos to 240
3920      sprite 3 to x_pos,y
3930        for x=1 to 25:next
3940    next
3950    sprite 3 off
3960    sprite 0 off :explode :explode :explode
3970    wait(30)
3980    sound 0,639,128
3990    text "Game Over..."dim 2 colour 29 to 70,60
4000    sound 1,0,32
4010    sound 1,507,96
4020    wait(32)
4030    text "Clean jumps:"dim 1 colour 19 to 105,90
4040    text str$(clean_pt) dim 1 colour 19 to 214,90
4050    sound 2,0,32
4060    sound 2,426,64
4070    wait(32)
4080    text "Damage:"dim 1 colour 19 to 105,100
4090    text str$(hit_pt) dim 1 colour 19 to 214,100
4100    wait(96)
4110    sound 0,639,4
4120    text "Jumps:"dim 1 colour 19 to 105,115
4130    text str$(jmp_cntr) dim 1 colour 19 to 214,115
4140    if jmp_cntr=clean_pt&jmp_cntr<>0
4150      text "PERFECT!"dim 1 colour 19 to 240,90
4160    else
4170      if jmp_cntr=0
4180        percent$="(0%)"
4190      else
4200        percent$=str$(clean_pt/jmp_cntr)
4210        percent$="("+mid$(percent$,3,2)+"%)"
4220      endif
4230    endif
4240    wait(5)
```

```
4250    text percent$ dim 1 colour 19 to 240,90
4260    sound 0,0,1
4270    sound 0,507,4
4280    text "Walls:"dim 1 colour 19 to 105,125
4290    text str$(wall_cntr) dim 1 colour 19 to 214,125
4300    sound 0,0,1
4310    sound 0,426,4
4320    sound 0,0,1
4330    sound 0,639,4
4340    sound 0,0,1
4350    wait(10)
4360    tot_score$=str$(clean_pt*15-hit_pt+(wall_cntr*2))
4370    text "Total score: "+tot_score$ dim 2 colour 29 to 40,145
4380  endproc
4500  proc check_for_clean()
4510    if hit(0,2)
4520      log_hit()
4530    else
4540      if side=0
4550        if x_pos>=hurdle_x
4560          clean_jump()
4570        endif
4580      else
4590        if x_pos<hurdle_x
4600          clean_jump()
4610        endif
4620      endif
4630    endif
4640  endproc
4700  proc jump_right()
4710    sprite 0 image 1
4720    for n=0 to 14
4730      x_pos=x_pos+jumprx(n)
4740      y_pos=y_pos+jumpry(n)
4750      if x_pos>=r_edge
4760        rev=true
4770        wall_cntr=wall_cntr+1
4780      endif
4790      if rev=true
4800        x_pos=x_pos-2*jumprx(n)
4810      endif
4820      sprite 0 to x_pos,y_pos
4830      if n>7
4840        sprite 4 to x_pos,y_pos-3-7
4850        sprite 3 to x_pos-1,y_pos-1
4860      else
4870        sprite 4 to x_pos,y_pos-7
4880        sprite 3 to x_pos-1,y_pos+1
4890      endif
4900      if playing(0)=0 then load_phrase()
4910      wait(2)
4920      move_hurdle(int(hurdle_speed/3))
4930    next
4940    sprite 0 image 0
4950    sprite 3 to x_pos-1,y_pos+1
4960    sprite 4 to x_pos,y_pos-7
4970  endproc
```

```
5000  proc jump_left()
5010    sprite 0 image 2
5020    for n=0 to 14
5030      x_pos=x_pos+jumplx(n)
5040      y_pos=y_pos+jumply(n)
5050      if x_pos<=l_edge
5060        rev=true
5070        wall_cntr=wall_cntr+1
5080      endif
5090      if rev=true
5100        x_pos=x_pos+2*abs(jumprx(n))
5110      endif
5120      sprite 0 to x_pos,y_pos
5130      if n>7
5140        sprite 4 to x_pos,y_pos-3-7
5150        sprite 3 to x_pos+1,y_pos-1
5160      else
5170        sprite 4 to x_pos,y_pos-7
5180        sprite 3 to x_pos+1,y_pos+1
5190      endif
5200      if playing(0)=0 then load_phrase()
5210      wait(2)
5220      move_hurdle(int(hurdle_speed/3))
5230    next
5240    sprite 0 image 3
5250    sprite 3 to x_pos+1,y_pos+1
5260    sprite 4 to x_pos,y_pos-7
5270  endproc
6000  proc instruct()
6010    cls :print :print
6020    print "Halloween Jack - beat your friend or sister!"
6030    print "--------------":print
6040    print " Has nothing to do with Halloween and his name is not Jack,"
6050    print " but he will jump if you press the letter 'L'.":print :print
6060    print "  - use 'Z' and 'X' keys to move left or right"
6070    print "    before time runs out.":print :print "  - accumulate points:"
6080    print :print "     15 - 'clean' jump over hurdle"
6090    print "      2 - bouncing off edge of screen":print :print
6100    print "     If you stumble on the hurdle, it will cost you !!":print
6110    print " Of course it's just for fun and meant to be a programming"
6120    print " example to demonstrate SuperBASIC features and techniques"
6130    print " such as sprite animation, multi-channel sound, & more."
6140    print :print :print chr$(9);chr$(9);"Press RETURN to begin..."
6150    while inkey()<>13:wend :print chr$(16);chr$(9);chr$(9);
6160    print "                        ":for x=1 to 27:print chr$(16);:next
6170  endproc
12000 data ".bcdefg."
12001 data "ijklmnop"
12002 data "q__tu__x"
12003 data "1_.45_.8"
12004 data "!@#$%&*("
12005 data "A_CDEFGH"
12006 data "IJ____OP"
12007 data ".RSTUVW."
12008 :
12100 data ".bcdefg."
12101 data "ijklmnop"
12102 data "q_.tu_.x"
12103 data "1__45__8"
12104 data "!@#$%&*("
12105 data "A_CDEFGH"
12106 data "IJ____OP"
12107 data ".RSTUVW."
12200 data ".bcdefg."
12201 data "ijklmnop"
12202 data "q._tu._x"
12203 data "1__45__8"
12204 data "!@#$%&*("
12205 data "ABCDEF_H"
12206 data "IJ____OP"
12207 data ".RSTUVW."
12208 :
12300 data ".bcdefg."
12301 data "ijklmnop"
12302 data "q__tu__x"
12303 data "1._45._8"
12304 data "!@#$%&*("
12305 data "ABCDEF_H"
12306 data "IJ____OP"
12307 data ".RSTUVW."
12400 data "..x....."
12401 data "..x^...."
12402 data "..x.x..."
12403 data ".xx..^.."
12404 data "......x."
12405 data "......x."
12406 data "......x."
12407 data ".....xx."
12408 :
12500 data "........"
12501 data "........"
12502 data "........"
12503 data "........"
12504 data "........"
12505 data "........"
12506 data "........"
12507 data "........"
```

```
12600 data ".~~~~~~."      12800 data "...{...."
12601 data "~~~~~~~~"      12801 data "...|...."
12602 data "~..~~..~"      12802 data "..}....."
12603 data "~..~~..~"      12803 data "...}...."
12604 data "~~~~~~~~"      12804 data "....}..."
12605 data "~~....~~"      12805 data "..~~~~.."
12606 data "~.~~~~.~"      12806 data "..~~~~.."
12607 data ".~~~~~~."      12807 data "........"
12608 :
12700 data "....{..."
12701 data "....|..."
12702 data ".....}.."
12703 data "....}..."
12704 data "...}...."
12705 data "..~~~~.."
12706 data "..~~~~.."
12707 data "........"
12708 :
13000 data "...{{{{{{{{............."
13010 data "..{{{{{{{{............."
13020 data ".{{{{{{{{.}............."
13030 data "{{{{{{{{.}}............."
13040 data "........}}}............."
13050 data "|||||.||}}}............."
13060 data "|||||.||}}}............."
13070 data "|||||.||}}............."
13080 data "|||||.||}.}............."
13090 data "........}}............."
13100 data "||.||||||}}............."
13110 data "||.||||||}}............."
13120 data "||.||||||}............."
13130 data "||.|||||}.}............."
13140 data "........}}............."
13150 data "|||||.||}}}............."
13160 data "|||||.||}}}............."
13170 data "|||||.||}}............."
13180 data "|||||.||}.}............."
13190 data "........}}............."
13200 data "||.||||||}}............."
13210 data "||.||||||}}............."
13220 data "||.||||||}............."
13230 data "||.|||||}............."
13240 :
14100 data 0,-1,1,-2,1,-2,1,-2,1,-1,1,-1,1,0,1,0,1,0,1,1,1,1,1,2,1,2,1,2,0,1
14110 data 0,-1,-1,-2,-1,-2,-1,-2,-1,-1,-1,-1,-1,0,-1,0,-1,0,-1,1,-1,1,-1,2
14120 data -1,2,-1,2,0,1
14130 :
15000 data 213,16,0,32,169,16,0,32,142,16,0,16
15010 data 213,16,0,32,169,16,0,32,142,16,0,16
15020 data 160,16,0,32,127,16,0,32,107,16,0,16
15030 data 160,16,0,32,127,16,0,32,107,16,0,16
15040 data 142,16,0,32,113,16,0,32,95,16,0,16
15050 data 107,8,0,24,127,8,0,8,142,16,0,32
15060 data 160,16,0,16,170,8,0,8,170,8,0,8,170,8,0,8,170,8,0,8
15070 data 190,8,0,8,190,8,0,8,190,8,0,8,190,8,0,8
```