DuetHero

A game in which you and your VIC-20 play Bach music together.

Author:	David Youd
8-bit system:	Commodore VIC-20
Language:	Commodore BASIC V2.0
Category:	PUR-120 (max 120 chars per line, BASIC abbreviations allowed)

I authorize the *BASIC 10 Liner Contest* judges the unrestricted publication rights over this program and its associated documentation and media files.

Links:

- Game: http://youdzone.com/c64/duethero
- Disk image containing the game: http://youdzone.com/c64/duethero.d64
- Youtube game playthrough: https://youtu.be/1MsPcFrKA6E
- This document hosted at http://youdzone.com/c64/duethero.pdf
    - Last documentation revision: March 14th, 2022
- Animated GIF preview: https://youdzone.com/c64/duethero.gif
- Competition games list: https://gkanold.wixsite.com/homeputerium/games-list-2022

Setup:

For play on a real VIC-20

Write the program to a floppy disk (e.g., with a ZoomFloppy setup) or use a modern VIC-20-connected storage device (e.g., SD2IEC).

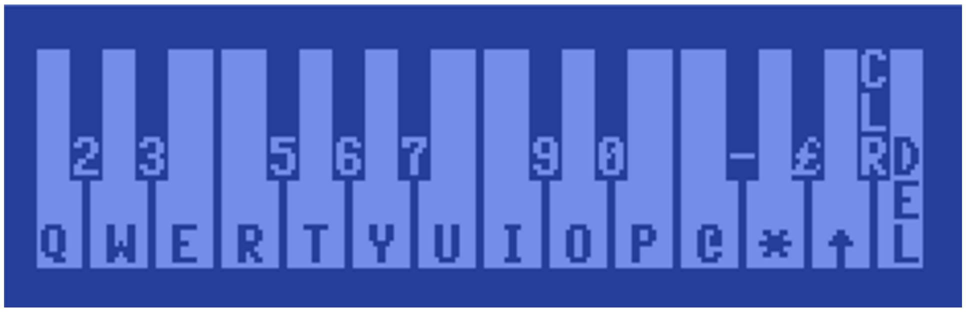
For the COMMODORE 64 with disk drive
INCREDIBLE MUSICAL KEYBOARD
SIGHT&SOUND
Transforms your Commodore 64 into an amazing musical instrument.
Play exciting music within minutes even if you've never played music before.
Create piano, guitar, synthesizer and electronic sounds with the accompanying music software program.
commodore VIC 20

If you have a Sight&Sound piano keyboard overlay, put it over your VIC-20's keyboard. If not, you can use the key input instructions detailed in the emulated play section below.

For emulated play

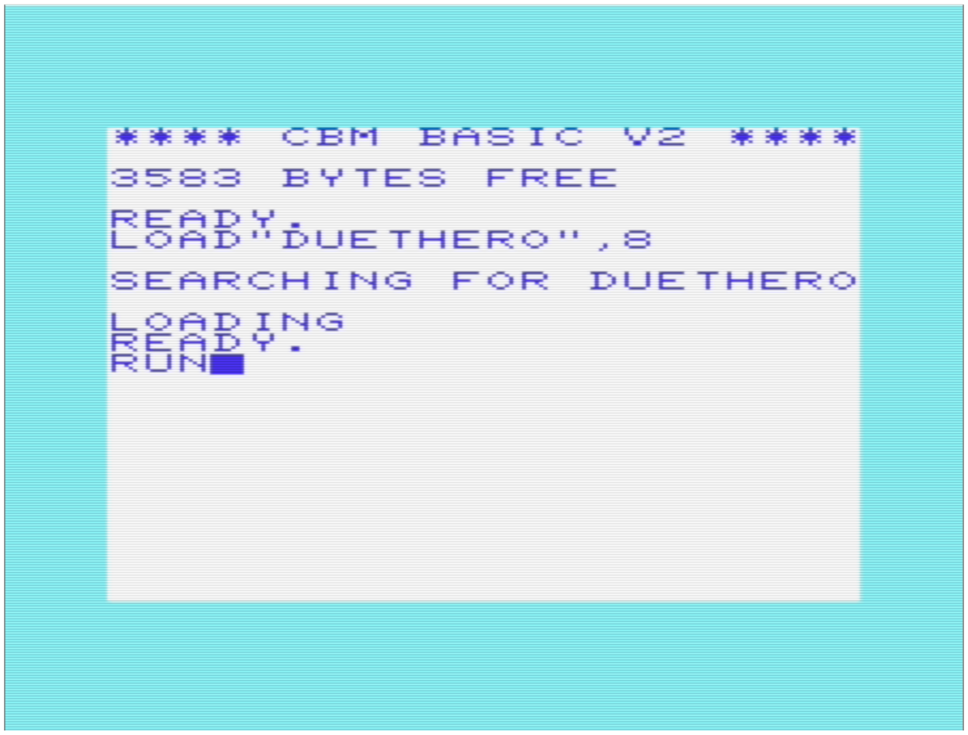
Using the VICE emulator (https://sourceforge.net/projects/vice-emu/), start a VIC-20 emulator (e.g., GTK3VICE-3.6.1-win64\bin\xvic.exe).

Since the piano keyboard overlay is not available for modern computer keyboards, use this note-to-keyboard mapping:



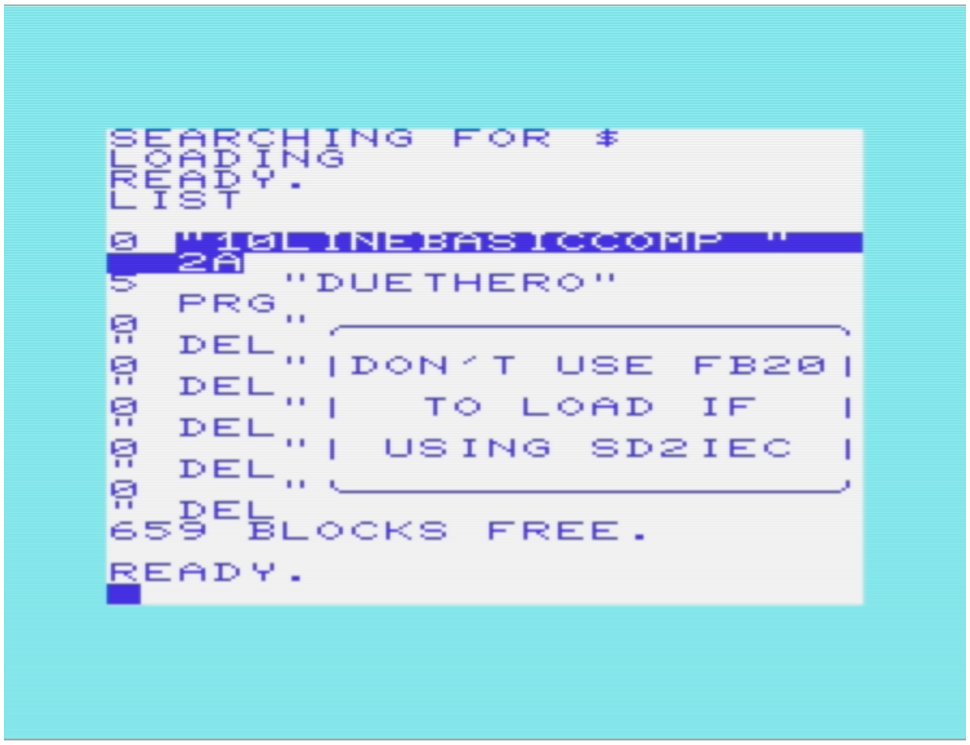
Starting the game

Load the game and run it.



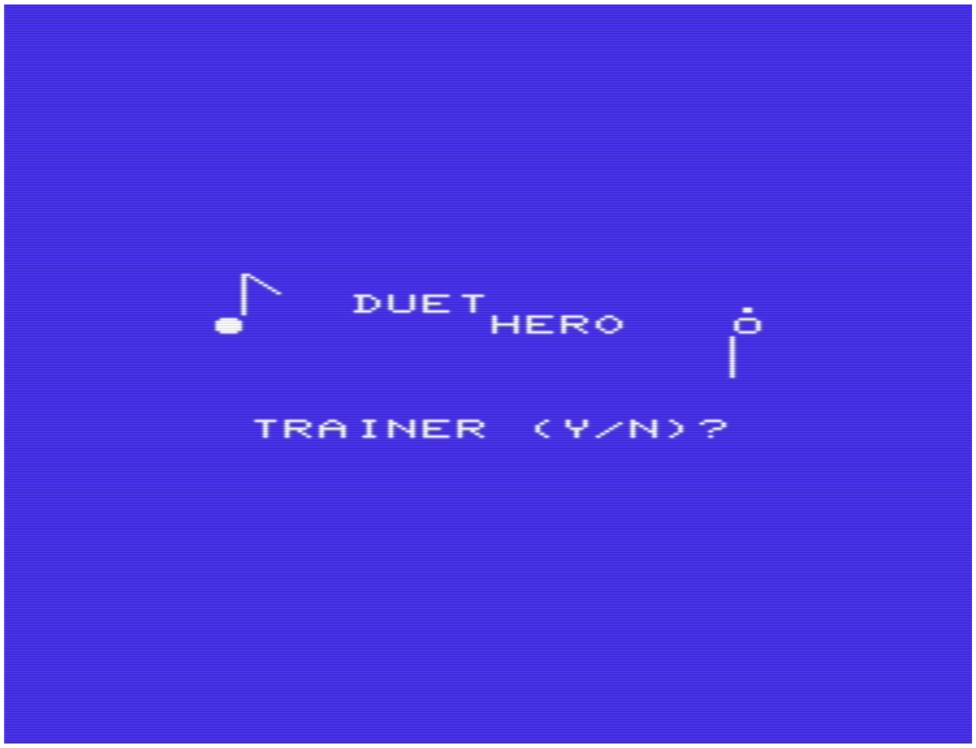
```
**** CBM BASIC V2 ****
3583 BYTES FREE
READY.
LOAD"DUETHERO",8
SEARCHING FOR DUETHERO
LOADING
READY.
RUN
```

Important: if using SD2IEC on real hardware, do not load the game from the “FB20” program selection menu, as that can corrupt the game such that you lose instantly. The directory listing contains a reminder of this warning:



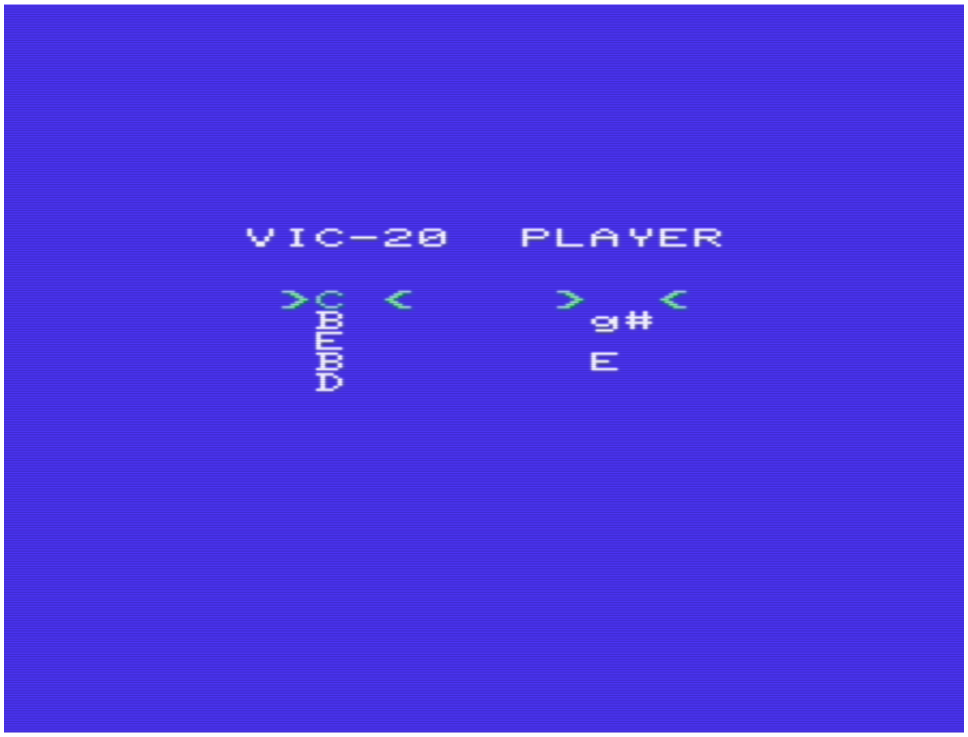
```
SEARCHING FOR $
LOADING
READY.
LIST
0 "10LINEBASICCOMP  "
  2A
5        "DUETHERO"
  PRG,"
0        ,-------------.
  DEL,"  |DON'T USE FB20|
0
  DEL,"  |   TO LOAD IF   |
0
  DEL,"  | USING SD2IEC   |
0
  DEL,"  '--------------'
0
  DEL
659 BLOCKS FREE.
READY.
```

A “trainer” mode can be selected by pressing “Y” on the game’s title screen. In this mode, you can practice the entire song without the song stopping early (which happens when a required note is not pressed at the right time).



Gameplay

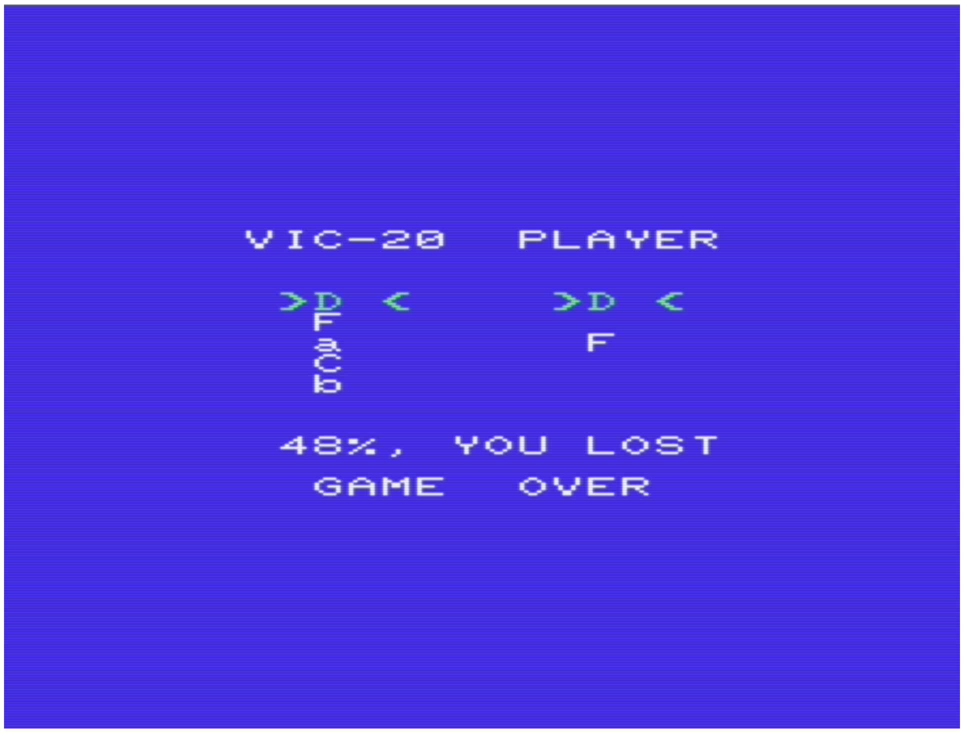
In DuetHero, you and your VIC-20 will create music together. The song is Bach’s Two Part Invention #13 (BWV 784), which is the tune that the Commodore company used in many of their TV commercials (e.g., https://www.youtube.com/watch?v=Q7_j_ABrkn8).



Musical notes will begin moving up the screen in two columns. Notes written in lowercase are for the lower octave, upper case for the upper octave. If the note is “sharp” (a black key on the piano keyboard overlay), it will be followed by a “#” symbol (see the “g#” in the above screenshot).

Notes are ready to be played when they enter the green area at the top of each column. The VIC-20 (left note column) plays the bass line automatically, while the player (right note column) must play the treble notes in time with the bass. Wrong notes are tolerated, as long as required notes are played. Unless the game is in training mode, failure to play the correct note when required will end the game.

(Note, if using real hardware and a Sight&Sound piano keys overlay, avoid pressing the very highest note, as it’s not needed to complete the game, and some of the piano overlays press the RESTORE key along with the INST DEL key at the same time. Occasionally, this event will cause the game to abort.)

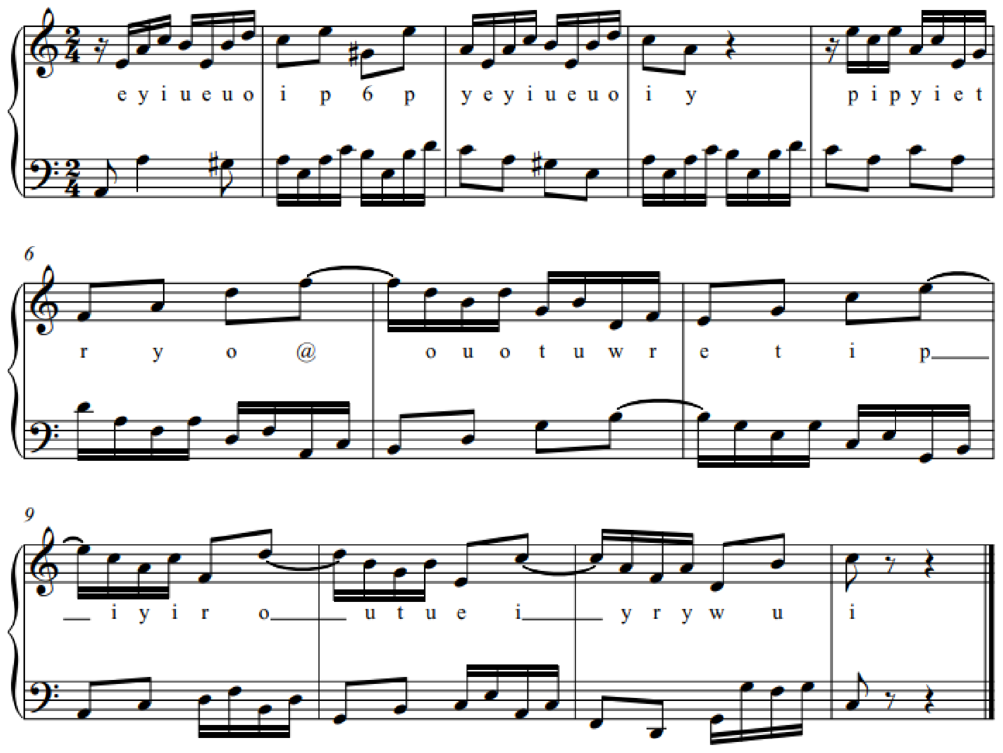


```
VIC-20   PLAYER

 >D <     >D <
  F
  a         F
  C
  b

 48%, YOU LOST
  GAME   OVER
```

A percent completion will be displayed when the game is over. Press any key to return to the game's title screen.

If you don't have the piano keyboard overlay…

Here are the keystrokes needed to win the game, just be sure to press them at the right time. If you know the bass line notes and rhythm, you won't even have to look at the screen. ;)



Program Description

This program is written in 10 lines of BASIC. It works on an unexpanded VIC-20. When running, the program uses 1744 bytes of RAM in the BASIC area (code and variables), and 245 bytes of RAM outside of the BASIC area (673 to 724 and 829 to 1022) to store data.

Commentary on each line below:

```
0 poke648,2:print"{home}{down*7}{right*7}Q2W3ER5T6Y7UI9O0P
{sh asterisk}-*{cm -}{pi}st@{reverse on}gosw
{arrow left}#'/37;?CGIKOQTWY{sh +}{sh -}{cm asterisk}{cm k}
{cm t}{cm @}":dimk%(255):fori=1to24:k%(peek(672+i))=i:next
```

Line 0:

- Location 648 is used to change the top page of screen memory. By changing this value, data can be stored directly into a chosen memory range using PRINT statements. This is not self-modifying code, but rather, a replacement for a READ/POKE loop over DATA statement values (for more details on this technique, see p56-57 of *Mapping the Commodore 64*, COMPUTE! Publications, 1984). In this case, POKE 648,2, followed by a cursor home, down 7, then right 7 is printing values at the start of some free non-basic memory at 673; 673 = 2(page)*256 + 7(down)*22(row width) + 7(right).
- Special PETSCII characters are represented in this code listing using CBM Prg Studio's (https://www.ajordison.co.uk/) syntax: {home}=19, {down}=17, {right}=29, {sh

asterisk}=96, {cm -}=220, {pi}=255, {reverse on}=18, {arrow left}=95, {sh +}=219, {sh -}=221, {cm asterisk}=223, {cm k}=161, {cm t}=163, and {cm @}=164. Note: {reverse on} sets bit 7 on screen codes.

- The PETSCII in the PRINT statements are not selected for their PETSCII values, but the values they'll have as screen codes once printed; example: to get the value of 31 in memory, "{arrow left}" is used (PETSCII value 95), which will have screen code value 31 when printed to memory. This can be confusing (certainly to me, since I have dyslexia), so I wrote a python program that converts arbitrary bytes for memory ranges into Commodore BASIC PRINT statements, using the special-character conventions from CBM Prg Studio (my development environment).
- The PRINT statement contains data for mapping the two piano octaves (note number 1 to note number 24) to the following commodore keys' PETSCII values: 81, 50, 87, 51, 69, 82, 53, 84, 54, 89, 55, 85, 73, 57, 79, 48, 80, 64, 45, 42, 92, 94, 19, and 20. It then contains a mapping from ascending note numbers to their pitch frequencies: 0 (no sound), 135, 143, 147, 151, 159, 163, 167, 175, 179, 183, 187, 191, 195, 199, 201, 203, 207, 209, 212, 215, 217, 219, 221, 223, 225, 227, and 228. The frequencies array is used by both the soprano oscillator (controlled by player) and the bass oscillator (controlled by VIC-20), even though the soprano and bass oscillators' pitches are offset by two octaves.
- The array K% is defined such that it can convert PETSCII values (0 to 255) from the keyboard buffer into note numbers (0 to 24, where 0 is note off). K% is sparsely populated (90% zeros), and these unmapped PETSCII-index keyboard assignments become note-off events.

```
1 poke648,3:print"{home}{down*2}"tab(17)"@ejmlelom@q@i@q@jejmlelom@
j@@@@@@qmqjmehf@j@o@r@@olohlcfe@h@m@q@@mjmf@o@@lhle@m@@jfjc@l@m@@@@
@@@";
```

Line 1:

- Stores the treble note data (and three bytes of bass data) in memory, starting at 829 (as 3*256 + 2*22 + 17) . This time TAB() is used to move the cursor right (more compact than 17 cursor-right control characters).
- All notes are staccato, and note duration is represented by varying the number of note-off (zero-value) events.
- Treble note data: 0, 0, 0, 0, 0, 5, 10, 13, 12, 5, 12, 15, 13, 0, 17, 0, 9, 0, 17, 0, 10, 5, 10, 13, 12, 5, 12, 15, 13, 0, 10, 0, 0, 0, 0, 0, 0, 17, 13, 17, 10, 13, 5, 8, 6, 0, 10, 0, 15, 0, 18, 0, 0, 15, 12, 15, 8, 12, 3, 6, 5, 0, 8, 0, 13, 0, 17, 0, 0, 13, 10, 13, 6, 0, 15, 0, 0, 12, 8, 12, 5, 0, 13, 0, 0, 10, 6, 10, 3, 0, 12, 0, 13, 0, 0, 0, 0

```
2 print"@j@v@@@u@vqvyxqx[y@v@u@q@vqvyxqx[y@v@y@v@[vrvorjml@o@t@x@@t
qtmqhlj@m@orloh@l@mqjmf@c@htrtm@@@@":poke648,30:poke649,1
```

Line 2:

- Bass note data (some on previous line): 0, 0, 0, 0, 10, 0, 22, 0, 0, 0, 21, 0, 22, 17, 22, 25, 24, 17, 24, 27, 25, 0, 22, 0, 21, 0, 17, 0, 22, 17, 22, 25, 24, 17, 24, 27, 25, 0, 22, 0, 25, 0, 22, 0, 27, 22, 18, 22, 15, 18, 10, 13, 12, 0, 15, 0, 20, 0, 24, 0, 0, 20, 17, 20, 13, 17,

8, 12, 10, 0, 13, 0, 15, 18, 12, 15, 8, 0, 12, 0, 13, 17, 10, 13, 6, 0, 3, 0, 8, 20, 18, 20, 13, 0, 0, 0, 0
- Return the top of screen memory to its default location.
- Limit the keyboard buffer to a single character capacity.

```
3 n$="  c c#d d#e f f#g g#a a#b C C#D D#E F F#G G#A A#B C C#D ":
print"{clear}{142}{white}{down*7}{right*3}{cm m}M{down}{left*2}
{cm m}{down}{left}Q{up}{right*3}duet{down}hero{up}{right*3}.{down}
{left}W{down}{left}{cm g}{down}{left}{cm g}{down*2}"
```

Line 3:
- N$ supports mapping from note values (0 to 24) to a two-character note name. Lowercase denotes the lower piano octave, and upper case denotes the upper octave.
- The PRINT statement draws the DuetHero game title screen. More CBM Prg Studio's syntax values: {clear}=147, {142}=142, {white}=5, {cm m}=167, {left}=157, {up}=145, and {cm g}=165. {cm m} changes to the upper-case character set when printed.

```
4 v=36878:pokev+1,110:print"    trainer (y/n)?":poke198,0:wait198,1
:gett$:print"{clear}{ct n}{down*5}{right*4}VIC-20  PLAYER":print"
{down*2}{right*5}{green}>  <    >  <{white}{down*4}"
```

Line 4:
- Change the screen and border color to blue.
- Ask player if they want trainer mode.
- Clear the keyboard buffer, wait until a key is pressed, and put that value in T$. If the value is "Y", then trainer mode is on, and the game will be unlosable.
- Draw the play field, one column for the VIC-20's bass note scroll, and the other column for the player's treble note scroll. {ct n} switches to the mixed-case character set. The angle brackets denote the top of the vertical note scrolls. When notes scroll into this position, it is during that window of time that the VIC20 plays the bass note, and the player attempts to perform the correct treble note.

```
5 pokev,15:a$="LOST":deffnv(x)=peek(o-(x>0)*x):fori=1to95:o=925:q=
fnv(i-4):m=fnv(i):o=828:r=fnv(i-4):n=fnv(i)
```

Line 5:
- Turn on the volume
- Game result string will contain "LOST" (unless later changed)
- Define the function FN V(X) to return "PEEK(MIN(O, O+X))". This works because Commodore BASIC V2 treats True as -1, False as 0, and because unary negation has higher operator precedence than multiplication. Therefore, -(X>0)*X is X when X>0, and 0 when X<=0. This function is needed because a note is played -4 steps from the note most recently displayed in the vertical scroll, but we don't want those initial negative indices underrunning the start of the note data.
- Setup I loop to process 95 16th note durations (the shortest rhythmic division in this game)

- Set memory offset (var O) for start of bass note data, then Q gets the next bass note to play. M gets the next bass note to display. Same is done for the treble, with N as the next note to display, and R as the note the player will attempt to perform. Sound lags five (as -4) steps behind the display in the scroll.

```
6 forj=7862to7950step22:pokej,peek(j+22):pokej+1,peek(j+23):pokej+8
,peek(j+30):pokej+9,peek(j+31):next:pokev-4,peek(697+q):pokev-2,0:
print"{up}{right*6}";
```

Line 6:

- The J loop scrolls both of the two-character note columns upward by copying screen memory from lower rows to higher rows.
- Play the bass note (note number in Q).
- Turn off the player's note before entering the next note duration window. Note: back when I had 11 BASIC lines, I was able to allow a note to be played in advance of the note's window if the previous note was already performed correctly (W was set to zero when correct note played, but then could be set to a new note value for the next upcoming note window). But I gave that up when getting it down to 10 lines (some hints of this previous logic remain on line 7). So now you have to play a note during the note window, which makes the game a bit more difficult. There's probably a 10-line solution for my previous approach, but I've not found it.
- The trailing semicolon allows the PRINT to be split across two lines.

```
7 printmid$(n$,m*2+1,2)"{right*6}"mid$(n$,n*2+1,2):fort=1to7:l=peek
(631):poke198,0:poke631,0:w=-(l=0)*w-(l>0)*k%(l):ifw>0thenpokev-2,
peek(697+w)
```

Line 7:

- PRINT continues by adding the next bass and treble note names to the bottoms of their respective vertical scrolls.
- Set up the T loop for 7 iterations. The player gets a window of 7 chances to enter the correct note.
- The player's input is read from the keyboard buffer and put into variable L. L is the key's PETSCII value, or 0 if no key pressed. After the read, the buffer is marked as empty, and the buffer contents set to 0.
- W=-(L=0)*W-(L>0)*K%(L) means if L is 0, then W is unchanged, but if L>0 then W gets the note number by using L's PETSCII value as an index into K%.
- If W's note number > 0, then play it (note: it doesn't hurt to re-play the currently playing note).

```
8 s=-(s=1)-(s=0and(r=0orr=w)):w=-(s=0)*w:next:on-(t$<>"y"ands=0and
r>0)goto9:s=0:next:a$="WON!":rem pbzvat fbba gb unccyrtnzrf
```

Line 8:

- S is 0 if the correct note has not yet been entered, and S is 1 if it was.
  S=-(S=1)-(S=0AND(R=0ORR=W)) means if S=1, no change to S. But if S=0 and R

(correct note) is either 0 (no note) or W (the user-selected note) then S=1 otherwise S=0.

- W=-(S=0)*W means that if S=0 then W is unchanged, but if S=1 then W=0
- This ends the definition of the T loop (that started in line 7); the opportunity to enter the correct note is over.
- ON GOTO statements compute a value from 0 to n, and use this value to select from a list of line numbers (in this case, only one) it will GOTO. A zero value results in no GOTO. ON GOTO has the advantage of not prematurely terminating a line like an IF/GOTO construct would. This ON GOTO (the only GOTO in the entire program), stops the game if a note was required (was not a rest) and the correct note wasn't entered. In trainer mode, the GOTO will not be taken.
- If the GOTO wasn't taken, then reset S to zero for the next note window.
- Complete the I loop setup in line 5.
- All possible notes have now been performed successfully, so change A$ from "LOST" to "WON!"
- The REM is a shoutout to my good friend and CEO of HappleGames, Inc.

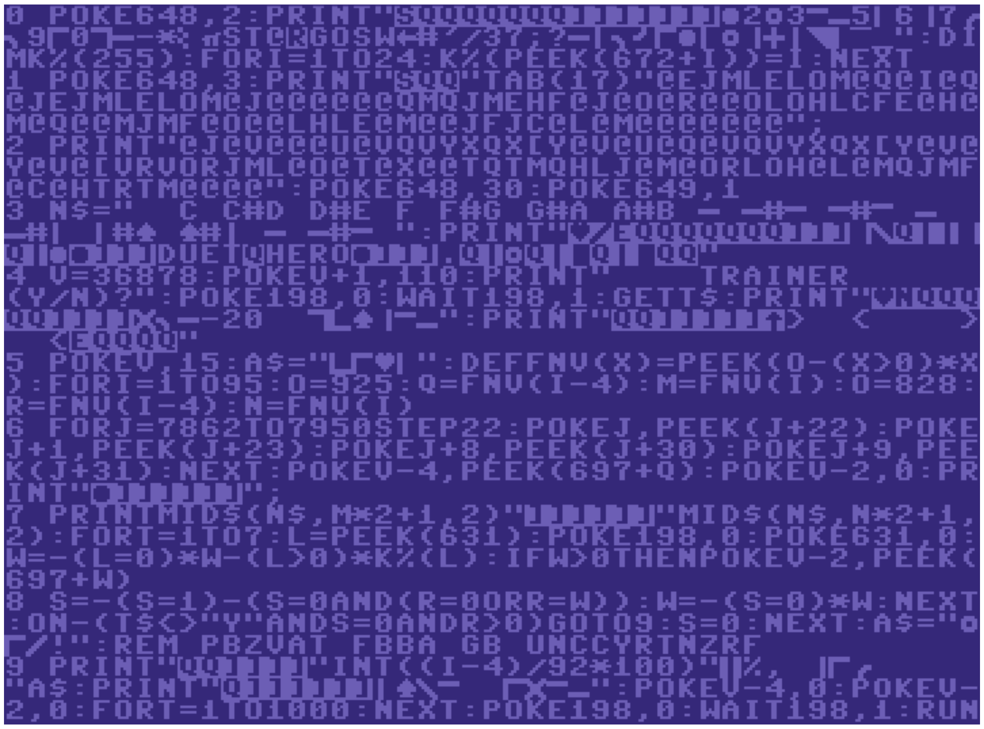
```
9 print"{down*2}{right*4}"int((i-4)/92*100)"{left}%, YOU "a$:print
"{down}{right*6}GAME  OVER":pokev-4,0:pokev-2,0:fort=1to1000:next:
poke198,0:wait198,1:run
```

Line 9:

- Print the percent game completion, and win or lose status, and display that the game is over.
- Turn off the oscillators.
- Insert a short delay, then clear the keyboard buffer, wait for a key to be pressed, and return to the game's title screen.

Code listing showing PETSCII control characters

This was generated using the DirMaster tool:



```
0 POKE648,2:PRINT"SQQQQQQQQ[illegible]●2○3[illegible]5[illegible]6[illegible]7[illegible]
[illegible]9[illegible]0[illegible]STC[illegible]GOSW←#'/37;?[illegible]":D[illegible]
MK%(255):FORI=1TO24:K%(PEEK(672+I))=I:NEXT
1 POKE648,3:PRINT"SQQ"TAB(17)"@EJMLELOM@Q@I@Q
@JEJMLELOM@J@@@@@@QMQJMEHF@J@O@R@@OLOHLCFE@H@
M@Q@@MJMF@O@@LHLE@M@@JFJC@L@M@@@@@@@";
2 PRINT"@J@V@@@U@VQVYXQX[Y@V@U@Q@VQVYXQX[Y@V@
Y@V@[VRVORJML@O@T@X@@TQTMQHLJ@M@ORLOH@L@MQJMF
@C@HTRTM@@@@":POKE648,30:POKE649,1
3 N$="  C C#D D#E F F#G G#A A#B [illegible]
[illegible]":PRINT"[illegible]
[illegible]DUET[illegible]HERO[illegible].[illegible]"
4 V=36878:POKEV+1,110:PRINT"     TRAINER
(Y/N)?":POKE198,0:WAIT198,1:GETT$:PRINT"[illegible]
[illegible]-20 [illegible]":PRINT"[illegible]>   <     >
  <EQQQQ"
5 POKEV,15:A$="[illegible]":DEFFNV(X)=PEEK(O-(X>0)*X
):FORI=1TO95:O=925:Q=FNV(I-4):M=FNV(I):O=828:
R=FNV(I-4):N=FNV(I)
6 FORJ=7862TO7950STEP22:POKEJ,PEEK(J+22):POKE
J+1,PEEK(J+23):POKEJ+8,PEEK(J+30):POKEJ+9,PEE
K(J+31):NEXT:POKEV-4,PEEK(697+Q):POKEV-2,0:PR
INT"[illegible]";
7 PRINTMID$(N$,M*2+1,2)"[illegible]"MID$(N$,N*2+1,
2):FORT=1TO7:L=PEEK(631):POKE198,0:POKE631,0:
W=-(L=0)*W-(L>0)*K%(L):IFW>0THENPOKEV-2,PEEK(
697+W)
8 S=-(S=1)-(S=0AND(R=0ORR=W)):W=-(S=0)*W:NEXT
:ON-(T$<>"Y"ANDS=0ANDR>0)GOTO9:S=0:NEXT:A$="[illegible]
[illegible]/!":REM PBZVAT FBBA GB UNCCYRTNZRF
9 PRINT"[illegible]"INT((I-4)/92*100)"[illegible]%, [illegible]
"A$:PRINT"[illegible]":POKEV-4,0:POKEV-
2,0:FORT=1TO1000:NEXT:POKE198,0:WAIT198,1:RUN
```

Proof of PUR-120 line-length compliance

This listing shows the BASIC abbreviations. All PETSCII graphic glyphs and control characters have been replaced with tildas ("~") for easier line-length validation (this does not change the line length):

```
Columns 1 to 60:
         111111111122222222223333333333444444444455555555556
123456789012345678901234567890123456789012345678901234567890

0pO648,2:?"~~~~~~~~~~~~~~~Q2W3ER5T6Y7UI9O0P~~-*~~st@~gosw~#'/
37;?CGIKOQTWY~~~~~~":dIk%(255):fOi=1to24:k%(pE(672+i))=i:nE

1pO648,3:?"~~~"tA17)"@ejmlelom@q@i@q@jejmlelom@j@@@@@@qmqjme
hf@j@o@r@@olohlcfe@h@m@q@@mjmf@o@@lhle@m@@jfjc@l@m@@@@@@@";

2?"@j@v@@@u@vqvyxqx[y@v@u@q@vqvyxqx[y@v@y@v@[vrvorjml@o@t@x@
@tqtmqhlj@m@orloh@l@mqjmf@c@htrtm@@@@":pO648,30:pO649,1

3n$="  c c#d d#e f f#g g#a a#b C C#D D#E F F#G G#A A#B C C#D
 ":?"~~~~~~~~~~~~~~M~~~~~~Q~~~~duet~hero~~~~~.~~W~~~~~~~~~"

4v=36878:pOv+1,110:?"    trainer (y/n)?":pO198,0:wA198,1:gEt
$:?"~~~~~~~~~~~VIC-20  PLAYER":?"~~~~~~~~~>  <    >  <~~~~~"
```

```
5pOv,15:a$="LOST":dEfnv(x)=pE(o-(x>0)*x):fOi=1to95:o=925:q=f
nv(i-4):m=fnv(i):o=828:r=fnv(i-4):n=fnv(i)

6fOj=7862to7950stE22:pOj,pE(j+22):pOj+1,pE(j+23):pOj+8,pE(j+
30):pOj+9,pE(j+31):nE:pOv-4,pE(697+q):pOv-2,0:?"~~~~~~~~";

7?mI(n$,m*2+1,2)"~~~~~~~"mI(n$,n*2+1,2):fOt=1to7:l=pE(631):pO
198,0:pO631,0:w=-(l=0)*w-(l>0)*k%(l):ifw>0tHpOv-2,pE(697+w)

8s=-(s=1)-(s=0aN(r=0orr=w)):w=-(s=0)*w:nE:on-(t$<>"y"aNs=0aN
r>0)gO9:s=0:nE:a$="WON!":rem pbzvat fbba gb unccyrtnzrf

9?"~~~~~~~"int((i-4)/92*100)"~%, YOU "a$:?"~~~~~~~~GAME  OVER"
:pOv-4,0:pOv-2,0:fOt=1to1000:nE:pO198,0:wA198,1:rU
```

Contact:

I had fun making this. If you wish, you can reach me at:

cryptoboy
at gmail
dot com