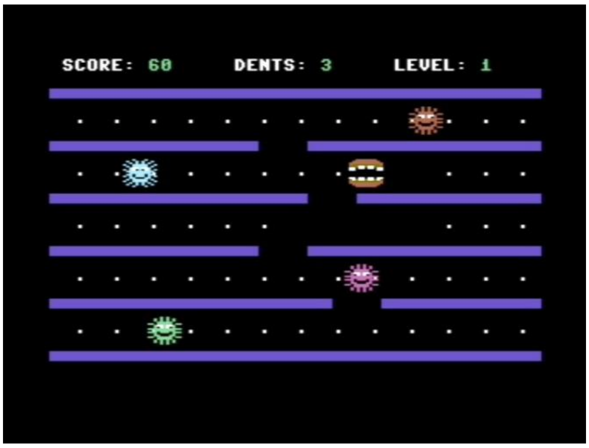
COVIDBREAKER

by Davide Fichera (Naufr4g0)

2020 BASIC 10-Liner contest
(category: EXTREM-256)



Game screenshot

SPECS:
Hardware: Commodore 64
Language: Standard BASIC V2

OBJECTIVE:

This is a classic game mainly inspired by JawBreaker, but with COVID viruses as enemies.
You can move your denture (main character) inside a maze made up of 5 tunnels one upon another. To pass from one tunnel to another you have to use the moving holes inside the walls.
The objective of the game is to clear the screen by eating all dots (65 per level) avoiding the viruses on the screen!
The speed of the enemies increases as the game progresses.
When you lost all your dentures the game ends and you have to press the fire button to play again.

Controls:
Use Joystick in port #2

CODE DESCRIPTION:

In this document I report first the most important variables used in the game and later the whole game code rewritten by separating the individual instructions and arranging them on different lines.
Inside code explanation I voluntarily removed the DATA statements, which make the writing too heavy, but I brought them further down at the end of the document.

Important Variables:

x = Player X-Coord
y = Player Y-Coord
p = Memory location of the char under the Player sprite (1024..2023)
hx(i) = X-coord of the i-th hole in the wall
hy(i) = Y-coord of the i-th hole in the wall
hd(i) = Speed of the i-th hole in the wall
mx(i) = X-Coord of the i-th enemy
my(i) = Y-Coord of the i-th enemy
mr(i) = Row position of the i-th enemy [rows are numbered from 0 (top) to 4 (bottom)]
md(i) = Speed of the i-th enemy
b(i) = bit value of the i-th enemy (vic sprites #1 - #4)

GAME CODE

```
0                                                 INIT GAME

    def fnr(x) = int(rnd(1)*x)                  - Define a function to extract a
                                                  random number from 0 to x-1
    py = rnd(-ti)                               - Init random number generator
    dim x,y,p,                                  - Declare x,y,p as fast access vars
    hx(3),hy(3),hd(3),                          - Allocate space in array memory for
    mx(3),my(3),md(3),mr(3)                       arrays containing monsters and holes
                                                  in the wall physics
    py = 214                                    - Init some constants:
    v = 53248: v2 = v+2: v9 = v+16: vc = v+30   - VIC consts
    jo = 56320: jl = 4: jr = 8: jv = 3: jd = 2  - Joystick control consts
    yj = 32                                     - delta-Y between map rows (in pixels)
    p0 = 2040: pm = 2041                        - Player sprite&first enemy sprite
    sb = 248: sm = 250                            pointers

    d8 = .125: f = 255                          - 1/8 value const, max value of a byte
    b(0) = 2: b(1) = 4: b(2) = 8: b(3) = 16     - value of monster bits
    si = 54296                                  - SID const for Volume control
    ax = 23: bx = 328                           - X-coord limits for Player

1

    ml = 342                                    - X-coord right limit for enemies
    ei = 80                                     - A useful const
    do = 46                                     - Set dot screen code to 46 (.)
    li = 3                                      - Init player Lives
    bs = 8                                      - Minimum base speed for enemies
    lv = 1                                      - Init Level number
    hr$(0) = "{reverse on}{blue}{cm y*2}"       - Strings needed for moving holes
    hl$(1) = "{reverse on}{blue}{cm               printing purposes
    y*2}{reverse off}"
    poke v + 32, 0: poke v + 33, 0              - Set screen border and background
                                                  screen colors to 0 (black)

    for i = 0 to 255
       read a
       poke 15872 + i, a                        - Load sprites DATA into memory
    next

    poke p0, sb                                 - Init Player Sprite pointer
    poke v+37, 9: poke v+38, 1                  - Init auxiliary colors for multicolor
                                                  sprites
    poke v+39, 2                                - Init Player color
    poke v, x: poke v+1, y                      - Move sprite to starting position
    poke v+28, 63                               - Set all sprites to multicolor mode
    gosub 7                                     - GO SUBroutine 7 (init level)

2                                                 MAIN GAME LOOP

    i = i+1 and 3                               - Cycle ‘i’ counter var in range 0..3
    j = not peek(jo)                            - Read Joy port#2 (reverse bits)
    vx = (jandjr) - (jandjl)*2                  - Calc player X-speed according to joy
                                                  position
    l= x+vx>=ax and x+vx<=bx                    - Check if jaws inside screen limit
    x = x - vx*l                                - Change player X-coord only if it’s
                                                  inside screen limits
    poke v, x and f                             - Move player sprite
    poke v9, peek(v9) and 254 or -(x>f)         - Set 9th bit of player X-coord
                                                  according to x var value
    p = p - sgn(vx)*l                           - move the pointer to char under
                                                  sprite
    if j and jv then                            - Execute following instructions only
                                                  if joy move in vertical direction
       if peek(p-ei+ei*(jandjd)) = 32 then      - Execute following instructions only
                                                  if there’s an open hole above or
          vy = -yj + (jandjd)*yj                  below the player
          y = y + vy
          poke v+1, y                           - Move player sprite on Y-axis
          p = p + sgn(vy)*160
```

GAME CODE

3

CHECK FOR DOTS

```
If peek(p) = do then                           - Execute following instructions only
                                                 if char under player is a dot
   poke p, yj                                  - clear char under sprite
   poke si, 15 : poke si, 0                    - Play a simple sound
   sc = sc + 10                                - Add 10 to score
   de = de + 1                                 - Keep track of eaten dots
   print "{home}{reverse off}{green}"          - Update score on the HUD
   tab(7) sc

   if de = 65 then                             - Check if all dots were eaten

      de = 0                                   - Clear eaten dots counter
      lv = lv + 1                              - Increase game level
      bs = bs + 2                              - Increase base speed for enemies
      gosub 7                                  - GO SUBroutine 7 (refresh level)
      goto 2                                   - GO TO 2 (main game loop)
```

4

UPDATE SPRITES ANIMS & HOLE POSITION

```
poke pm+i, sm + (rand1)                        - Update enemy sprite (anim)
poke p0, sb + (r*2and1)                        - Update player sprite (anim)
poke py, hy(i) : print                         - Fast cursor moving to the row
                                                 corresponding to i-th hole
hd= -(hd(i) > .)                               - hd = 0 if hole is moving left
                                                    = 1 if hole is moving right
print tab(hx(i)) hl$(hd) "     " hr$(hd);      - Print hole in the i-th wall
hx(i) = hx(i) + hd(i)

if hx(i)>33 or hx(i)<. then                    - If the hole moves outside the screen
   hd(i) = -hd(i) : hx(i) = hx(i) + hd(i)        limits changes hole movement dir
```

5

UPDATE ENEMY POSITION

```
mx(i) = mx(i) + md(i)                          - Update enemy X-coord according to
                                                 its direction
poke v2 + i*2, mx(i) and f                     - Update enemy X-coord on screen
poke v9, peek(v9) and f-b(i) or                - Update 9th bit of X-coord
(b(i)*-(mx(i)>f))

if mx(i)<. or mx(i)>ml then                    - Check if enemy go outside screen
                                                 borders
   md(i) = abs(md(i))*sgn(rnd(1)-.5)           - Change randomly the direction
   a = mr(i)                                   - 'a' var was used to swap values
   mr(i) = hm                                    between hm and mr(i) vars
   my(i) = 77 + 32*hm                          - Set new y-Coord for enemy
   hm = a
   mx(i) = -ml*(md(i)<0)                       - Set new x-Coord according to dir
                                                 (all the way to the left or to the
                                                 right)
   poke v2+1+i*2, my(i)                        - Update Y-coord on screen
   du = peek(vc)                               - Dummy variable to avoid double
                                                 collision detection
```

6

COLLISION DETECTION

```
r = r + .25                                    - Increase variable for sprite anims
                                                 purposes
dc = peek(vc)                                  - Read hardware sprite collision reg.
on -((dcand1)=.) goto2                         - If player doesn't collide go to the
                                                 begin of the main loop
li = li -1                                     - ELSE (LOST LIFE): decrease lives
gosub 8                                        - GO to SUBroutine 8 (refresh screen
                                                 avoiding dot initing)
on -(li>0) goto 2                              - If lives>0 go to begin of main loop
poke v+21, 0                                   - ELSE (GAME OVER): Disable all
                                                 sprites on screen
```

```
poke py, 11: print
print tab(14) "{white}game over!"
wait jo, 127, 127
run
```

- Print “Game Over” at the center of the screen
- Wait for Joy Fire to be pressed
- Restart game

7

INIT GAME SCREEN SUBROUTINE

```
print "{blue}{clear}{white}";
```

- Clear scren

```
for i = 0 to 4
   poke py, i*4+3: print

   for j = 0 to 12
      print"{white}  .";
   next
next
```

- A FOR loop that draws 5 rows of 13 dots each.

8

- INIT MONSTERS & PLAYER START POSITIONS

```
poke v+21, 0
```

- Disable all sprites

```
for i = 0 to 5
   poke py, i*4+1: print
   print "{blue}{reverse
   on}{right}{delete}{cm
   y*39}{left}{148}{cm y}{reverse off}";
next
```

- A FOR loop that draws 6 walls

```
for i = 0 to 3
   hx(i) = 15
   hy(i) = i*4 + 5
   hd(i) = 4*((iand1)-.5)

   mr(i) = i - (i>1)

   md(i) =
   bs*(int(rnd(1)*2)+1)*sgn(rnd(1)-.5)
```

- FOR loop for enemies & holes init
- set hole x-coord to 15
- set hole y-coord
- set hole speed (alternates left and right)
- set row indicator (2 row stay free for player
- set enemy X-speed to random value (1x or 2x base enemy speed)

9

```
   my(i) = 77 + 32*mr(i)
   mx(i) = -ml*(md(i)<0)
   poke v+i*2+2, mx(i) and f
   poke v+i*2+3, my(i)
   poke pm+i, 250
next
```

- set enemy y-coord
- set enemy x-coord
- move sprite to X,Y
- set enemy sprite anim

```
poke v+21, 63
i = 3
x = 161
y = 141
p = 1522

poke v, x: poke v+1, y
du = peek(vc)

print "{home}{white} score:" tab(15)
      "dents:{green}" li; tab(28)
      "{white}level:{green}" lv

hm = 2
return
```

- Enable all sprites
- set i to 3
- set player.X
- set player.Y
- set screen position of char under player sprite
- Move player sprite to X,Y
- Dummy var to avoid unwanted collision detection
- Print HUD infos
- init empty row (w/o enemies)
- Return from subroutine

SPRITES DATA

Here are the sprites DATA used in the game and arranged in groups of 9 bytes per line. Originally DATA are placed at the end of some of the 10 lines of game code.

```
1000 REM JAWS #1
1010 DATA 0,0,0,0,0,0,2,170,160
1020 DATA 10,170,168,42,170,170,37,85,86
1030 DATA 21,247,213,60,243,207,60,0,15
1040 DATA 0,0,0,0,0,0,0,0,0
1050 DATA 0,0,0,60,243,207,29,247,221
1060 DATA 37,85,86,42,170,170,10,170,168
1070 DATA 2,170,160,0,0,0,0,0,0
1080 DATA 0
1090 REM JAWS #2
1100 DATA 0,0,0,0,0,0,0,0,0
1110 DATA 0,0,0,2,170,160,10,170,168
1120 DATA 42,170,170,37,85,86,21,247,213
1130 DATA 60,243,207,60,0,15,60,243,207
1140 DATA 21,247,213,37,85,86,42,170,170
1150 DATA 10,170,168,2,170,160,0,0,0
1160 DATA 0,0,0,0,0,0,0,0,0
1170 DATA 0
1180 REM COVID #1
1190 DATA 0,0,0,0,0,0,0,128,128
1200 DATA 8,34,8,2,34,32,32,170,130
1210 DATA 8,170,136,2,251,224,34,200,226
1220 DATA 10,170,168,2,170,160,10,42,40
1230 DATA 34,42,34,2,128,160,8,170,136
1240 DATA 32,170,130,2,34,32,8,34,8
1250 DATA 0,128,128,0,0,0,0,0,0
1260 DATA 0
1270 REM COVID #2
1280 DATA 0,0,0,0,0,0,0,34,0
1290 DATA 2,34,32,2,34,32,0,170,128
1300 DATA 40,170,138,2,251,224,2,59,32
1310 DATA 42,170,170,2,170,160,42,42,42
1320 DATA 2,0,32,2,128,160,40,170,138
1330 DATA 0,170,128,2,34,32,2,34,32
1340 DATA 0,34,0,0,0,0,0,0,0
1350 DATA 0
```