	"PURGE" By Eric Henneke 2021 BASIC 10-Liner PUR-80 Atari BASIC	0U=4^5:GR.8:GR.28:DIMH$(1),V$(33824-ADR(H$)),A$(U),F$(48):A=ADR(A$):F.I=A-7TOA-4 1POKEI,90:POKEI+U,90:N.I:READB,H$,A$:F$=A$:G=40:U=880:F$(32)="♥(66←←(♥●)CCCC)● 2V$=" ":V$(U)="":V$(2)=V$:T=T+6:F.L=0TO1:F.I=1TOT:R=RND(0)*878+2:I=I-V$(R,R)=" 3V$(R,R)=CHR$(G+L):N.I:N.L:V$(R,R)=H$:P=P+6+T/G:I=P-3:V=0:W=0:D=0:E=0:?#6;"↖";V$ 45E.0,T+12,8:T.9:C.35:PL.0,B:DR.P-1,B:M=M-G*D:N=N-24*E:POKE77,1:T.4:G.7:D.23,▯ 5POKE756,133+J:V=X:W=Y:50.0,P,Z<41,9:O=STICK(0):IF(O=13ORO=7ORO=14ORO=11)THEN5=O 6D=(5=7)-(5=11):E=(5=13)-(5=14):C=38-(5=14)*2+D:M=X+D:N=Y+E:IFZ=169THENP=P+3*P/I 7LOC.M,N,Z:IFZ-40THENC.C:X=M:Y=N:PL.X,Y:C.32:PL.V,W:IFZ>40THENL=L+1:IFL=T+2THEN2 8J=1+(J<4)*J:P=P-.03:PL.P,B:G.5:D.┻{fBBB♥/┏ ┻p┻┛/♥♥BBBf{┻♥●0┗━┛0●♥({←←66{♥●kCCK● 9GR.18:I=T/6:?#6;I*3*(I+1)-T+L-2:POS.18,0:?#6;I:F.I=0TO1:I=I-STRIG(0):N.I:RUN
L#	**Code**	**Documentation / Explanation**
0	`U=4^5:GR.8:GR.28:DIMH$(1),V$(33824-ADR(H$)),A$(U),F$(48)`	Calling Graphics 8 clears a huge swath of memory from $8100 to $9F87 which is a fast efficient way to initialize a lot of memory. Then Graphics 28 sets the actual mode used for the game (Graphics 12 with no text window). By setting DIM H$(1) as the very first string variable we can then do math from there to assure other strings are located at specific memory locations. Next V$ is Dimensioned as V$(33824-ADR(H$)) which assures that no matter where H$ is located, we know V$ will end at 33824($8420), and the next variable, A$, will always start at 33825($8421). And since A$ is 1024bytes it means F$ always starts at 34849($8821).
	`:A=ADR(A$):F.I=A-7TOA-4`	A is assigned the address of A$. For/next loop is started (described on next line)
1	`POKEI,90:POKEI+U,90:N.I:READB,H$,A$:F$=A$:G=40:U=880`	The logic inside this for/next populates 4 bytes starting at 33818($841A) with that value 90. Later we will redefine the character set by pointing to $8400, so by changing bytes at $841A we are redefining the 3rd character of the characterset(ATASCII 39). The loop also populates 4 bytes starting one page higher at 34832($881A) because we will be flipping between two character sets for animation during the main loop execution, but we want the 3rd character to be the same all of the time (because this character is used to draw the energy status bar). After the for/next loop completes, a Read is used to populate B and H$ with constant values from the Data on line 4, then A$ is loaded with a very long string of 46 ATASCII characters from the Data on Line 8 which will be used to redefine a portion of the character set. F$ is then assigned the value of A$ as well. Then G and U are assigned constant values.
	`:F$(32)="♥(66←←(♥●)CCCC)●`	F$(32) assigns a string of 16 characters starting at the 32nd bytes inside of F$, by doing this we leave the first 31 bytes as-is. The result is F$ now contains 47 bytes, with the first 31 being the same as A$ and the last 16 bytes being different than A$. This creates the differences in the two character sets that will be flipped between during the mainloop for the animation.
2	`V$=" ":V$(U)="":V$(2)=V$:T=T+6:F.L=0TO1:F.I=1TOT`	Line 2 initializes a new wave. First clear out all 880 bytes of V$. T is incremented by 6 (this controls how many Orbs will populate the wave. Then there are two nested For/next loops. The outer loop runs twice (once to place the Blockers and a second time to place the Orbs). The logic inside the For/Next loop described on next row…
	`:R=RND(0)*878+2:I=I-V$(R,R)="):I=I-V$(R,R)="`	The logic inside this for/next loop generates a random value (between 2 and 880) for each Orb that will be populated for the new wave. (See more on this logic in next line)
3	`V$(R,R)=CHR$(G+L):N.I:N.L:V$(R,R)=H$`	Then V$ is populated at each of those random locations characters representing Orbs or Blockers. The first time the loop runs L=0 and G is a constant value of 40, so CHR$(40) is populated in each V$ location (Char 40 is the Orbs) and the 2nd time the loop runs L=1 so CHR$(41) is populated in each V$ location (Char 41 is the Blockers). If that location of V$ is already populated with right_parenthesis (CHR$ 41) it means a Blocker was already placed there so the loop counter I is decremented by -1 which forces the loop to generate an additional random number by repeating that iteration of the for/next loop. This way the proper number of Orbs will be populated for the wave even if the random numbers occasionally land on a position already populated by a Blocker. When the nested for/next loops are completed, the last location populated at V$(R,R) is changed to H$ which has a constant string value of inverse_rigtht_parenthesis (CHR$ 169) which represents the special Pink Leader Orb.
	`:P=P+6+T/G:I=P-3:V=0:W=0:D=0:E=0`	P is incremented by 6 (this controls how much energy you start each wave with) as well as a calculation to give a little extra starting energy progressively with each higher wave. I is updated to P-3 which is used later during the main loop. Some other variables are set to zero to initialize.

	`:?#6;"K";V$`	The screen is cleared and V$ is printed. Since V$ is 880 bytes long, it covers the entire playfield, resulting in the locations of all the Orbs and Blockers showing up on the screen for the new wave.
4	`SE.0,T+12,8:T.9:C.35:PL.0,B:DR.P-1,B:M=M-G*D:N=N-24*E`	Line 4 does three things: 1) it changes the setcolor each wave, 2) it updates the energy status bar on the screen, and 3) it recalculates the player's X and Y coordinates (using temporary variables M and N, which are used later to update variables X and Y in Line 7) so the player will "wrap around" to the other side of the screen. The Trap 9 is used here as a way to test if the game is over. The reason this works is because the Plot/Drawto that follows displays the energy bar at the bottom of the screen. If energy gets dropped to <0 then it attempts to Plot a value that is off the left boundary of the screen which would normally cause an error, but the Trap catches the error and redirects the program flow to Line 9(the game over sequence). In this way, the Trap is effectively used as a controlled branch statement and only takes up 3 characters of code.
	`:POKE77,1:T.4:G.7:D.23,D`	Note there is also a Poke 77,1 here to prevent attract mode from ever kicking in (it is here mainly because it fit here code-wise! and also because this line is not executed every main loop so it isn't as impactful to performance). TRAP 4 then changes the line destination to branch to if an error happens later in the program flow(this is necessary for the logic on Lines 7 & 8 to work and it is explained more below on those lines). Finally, a Goto 7 directs the program flow to Line 7. The remaining space at the end of Line 4 holds a Data statement with values assigned to variable B and H$ during the Read on Line 1.
5	`POKE756,133+J:V=X:W=Y:SO.0,P,Z<41,9:O=STICK(0)`	This is the start of the main program loop. The first thing that happens is the Poke 756,133+J points to an alternating area of memory to redefine the character set. J will be a value between 1 and 4 (as determined in Line 8 later in the main loop). So it means 133+J will be a value between 134 and 137. Poking 756 with 134, or 135 results in pointing to the character data in page $8400. While Poking 756 with 136, or 137 results in pointing to the character data in page $8800. It means that every 3rd iteration of the main loop alternates the location for the character data. This creates the animated "pulsating Orb" effect by quickly changing character set data. Next V and W variables are updated with current X and Y values to store them temporarily. A Sound operation here sets a distorted tone if the last coordinate the player moved to was occupied by an Orb, otherwise it just generates silence, the frequency of the tone is also determined by P(energy level) which makes the pitch go up as energy gets lower and lower serving as an additional warning to the player. O is then assigned the value of joy Stick(0) as a temporary holder of the latest joystick movement.
	`:IF(O=13ORO=7ORO=14ORO=11)THENS=O`	The IF statement checks to see if a valid joystick move has been made (up, down, left or right) and if so then it assigns the value of O to variable S. S will be used to take action on the joystick movement during the rest of the main loop.
6	`D=(S=7)-(S=11):E=(S=13)-(S=14):C=38-(S=14)*2+D:M=X+D:N=Y+E`	On Line 6, first D and E are calculated using boolean math based on the joystick direction. D is set to either -1 or +1 or 0 depending on if joystick is pressed left or right or neither, and E is set to -1 or +1 or 0 depending on if it is up or down or neither. Variable C is used to set the Color when Plotting the player's "ship" on the screen(this happens on Line 7). By changing the value of C before the Plot of the player's ship this will determine which character to use (either the ship pointing up, down, left, or right) on the screen, since Graphics 12 is character-based. This code on Line 6 calculates and changes C to the appropriate value based on the joystick direction that is being pressed (which is held in variable S), resulting in C=36,37,38 or 39. Then D is combined with X value and stored in temp variable M. The same is done combining E with the Y value and stored in temp variable N. This way M and N now hold what will be the next location of the player's ship.
	`:IFZ=169THENP=P+3*P/I`	Here we check Z to see if the player has eliminated the Pink Leader Orb (which is defined as Character 169, which is an inverse_right_parenthesis). If Z=169, then this calculation increases the energy level P depending on how much energy has already been depleted during the wave. The sooner the player eliminates the Pink Orb, the more energy will be increased.

7	LOC.M,N,Z:IFZ-40THENC.C:X=M:Y=N:PL.X,Y:C.32:PL.V,W	This is where the Trap 4 statement near the end of Line 4 becomes important to the logic flow...that Trap 4 remains active during the main loop execution and serves as a controlled branch operation when an (intentional) Error will be forced if the player's coordinates move off the edge of the screen. The Locate is used to see what, if anything, is in the X,Y location the player is about to occupy and the value is assigned to Z. The IF is used to check Z to see if it is equal 40 which would mean the player has run into a Blocker (Blocker is represented by character 40 which is a left parenthesis). If Z is NOT 40, then Color is set to value C (which was determined on Line 6). Then the X and Y values are updated with from temp variables M and N (these are the coordinates of the player's ship) before Plotting the X,Y on the screen to display the Player ship in its new location. If X or Y is a value outside of the screen boundaries this would normally cause an error, but the Trap will instead redirect the program flow to Line 4, otherwise program flow continues. Color is set to 32 (which is a blank character on Graphics 12 screen) before Plotting V,W which will erase the old Player ship location.
	:IFZ>40THENL=L+1:IFL=T+2THEN2	If Z is > 40 it means that an Orb has been eliminated (regular Orbs are character 41 and Pink Orb is 169) so L is increased by +1 (L tracks how many Orbs have already been eliminated within the current wave). Finally, if L=T+2 it means all the Orbs have been eliminated and the next wave is initialized by branching to Line 2.
8	J=1+(J<4)*J:P=P-.03:PL.P,B:G.5	The value of J is updated based on it's current value. If J is <4 then it will be incremented by +1, but if J is already=4 then J will be set to 1. Therefore, each loop execution will increment J from 1,2,3,4 and then back to 1 (this is used in the Poke on Line 5 to point to the proper characterset data to animate the Orbs). P is the energy tracker and here it is decremented by 0.03, this makes the energy slowly deplete with each main loop iteration. Then the Plot P,B (where B is set to 23) will erase the end of the energy status bar at bottom of the screen (this is because Color is still set to 32 at this point from when it was set in Line 7). By decrementing P by 0.03 each loop, it means one "block" of the energy status bar goes away every 33 main loop iterations.
	:D.┴⟨fBBB♥/▀┴p┘/♥♥BBBf⟨┴♥♦0┴▀┴0♦♥(▪▪-666(♥♦kCCk♦	The remaining space at the end of Line 8 holds a Data statement with string data that is used to initialize A$, which is used in the characterset redefinition process.
9	GR.18:I=T/6:?#6;I*3*(I+1)-T+L-2:POS.18,0:?#6;I:F.I=0TO1	Line 9 is the END OF GAME sequence. Screen is cleared with a change to Graphics mode 18 (Graphics 2 with no text window) to display big/clear text on the screen. I is calculated as T/6 which represents the wave that the Player made it to before game over. Then the score (how many total Orbs were eliminated) is displayed in Graphics 2 font using a ? #6, along with the wave that the Player made it. To display the score, a convoluted calculation has to be done since the program only tracks the total Orbs eliminated during the current wave. So I*3*(I+1)-T+L-2 is a way of calculating the total number of Orbs eliminated duirng the entire game. And lastly, a For/Next loop is started, from 0 to 1, to wait for the Player to press the joystick trigger to start a new game.
	:I=I-STRIG(0):N.I:RUN	If the Trigger is NOT pressed, then I is decreased by -1. This way the For/Next loop will never end until the Player presses the trigger, which will allow I to be increased to 1 when that happens. When this happens, the for/next loop exits, and RUN starts the program over with a new game initialization.